UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

David Vidmar

Razvoj spletne in androidne aplikacije za sledenje položaja oddaljene mobilne naprave

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

David Vidmar

Razvoj spletne in androidne aplikacije za sledenje položaja oddaljene mobilne naprave

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Marko Privošnik

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Uporaba mobilnih naprav navadno poteka preko mobilnih aplikacij. Te so lahko razvite posebej za dano mobilno platformo ali pa temeljijo na spletnih tehnologijah in so zato širše uporabne. V okviru diplomske naloge preučite oba pristopa in ju uporabite na praktičnem primeru razvoja aplikacije za sledenje položaja oddaljene naprave s poudarkom na razvoju uporabniškega vmesnika. Na podlagi razvite rešitve, oba pristopa ovrednotite in ju primerjajte.

Zahvaljujem se mentorju viš. pred. dr. Marku Privošniku za svetovanje in pomoč pri izdelavi diplomske naloge.

Kazalo

Seznam kratic

kratica	angleško	Slovensko
GPS	Global Positioning System	Globalni sistem pozicioniranja
AVD	Android Virtual Machine	Virtualna naprava za Android
API	Application programming interface	Aplikacijski programski vmesnik
TCP	Transmission Control Protocol	Nadzorni protokol za prenos
IP	Internet protocol	Internetni protokol
HTML	Hyper Text Markup Language	Jezik za označevanje nadbesedila
CSS	Cascading Style Sheets	Kaskadne stilske podloge
XML	Extensible Markup Language	razširljivi označevalni jezik,
PHP	Personal Home Page Tools	orodja za osebno spletno stran
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
SDK	software development kit	Orodje za razvoj programske opreme

Povzetek

Naslov: Razvoj spletne in androidne aplikacije za sledenje položaja oddaljene mobilne naprave

To diplomsko delo obsega opis, izdelavo in primerjavo dveh različic enake aplikacije za izris geografskega položaja oddaljene mobilne naprave. Prva različica aplikacije je izdelana izključno za mobilne naprave s operacijskim sistemom Android, druga različica pa je spletna različica, ki je neodvisna od platforme. V prvem delu se osredotočamo predvsem na tehnologije in orodja, ki smo jih uporabljali pri izdelavi aplikacij. Opisali smo sistem GPS, aplikativni sistem Google Maps, spletne tehnologije, operacijski sistem Android, razvoj aplikacij za takšne sisteme ter razvojno orodje Android Studio. V drugem delu smo opisali načrtovanje in izdelavo naše aplikacije za določanje geografskega položaja oddaljene mobilne naprave. Aplikacija ima možnost oddajanja podatkov o našemu trenutnemu geografskemu položaju ter možnost prejemanja podatkov o geografskemu položaju neke druge mobilne naprave. Prejeti geografski položaj nam aplikacija izriše na zemljevidu. Na koncu smo obe različici aplikacije testirali in ju primerjali ter opisali naša opazovanja.

Ključne besede: mobilne naprave, spletne tehnologije, mobilna aplikacija, spletna aplikacija, Android, Google Maps, GPS.

Abstract

Title: Development of a web and an android application for tracking the position of a remote mobile device

This thesis covers a description, creation and comparison of two different versions of the same application for tracing the geographic position of a remote mobile device. The first version of the application is made exclusively for mobile devices with the operating system Android; the second version is an online version, which is independent of the platform. In the first part of thesis, we mainly focus on the technology and the tools that were used to create the applications. We described the GPS system, application system Google Maps, web technologies, operating system Android, the development of applications for such systems and the development tool Android Studio. In the second part, we described the design and creation of our application for determining a geographic position of a remote mobile device. The application has the ability to broadcast the information of our current geographical position and the ability to receive the data of geographical position of some other mobile device. The application draws the received geographical position on the map. In the last part, we tested both application versions, compared them and described our observations.

Keywords: mobile devices, web technologies, mobile application, web application, Android, Google Maps, GPS.

1 Uvod

Živimo v času mobilnih tehnologij, ki nam vsakodnevno olajšujejo življenje. Pametne telefone uporabljamo za veliko več kot samo telefonske pogovore in tekstovna sporočila. Uporabljamo jih tako za brskanje na spletu, kot igranje igric, prav tako pa nam pridejo priročno pri raznih poslovnih stvareh, saj obstaja veliko različnih aplikacij za vse naše potrebe. Pametni telefoni tako zavzemajo vse več in več različnih funkcionalnosti, ki so jih do sedaj nudili računalniki, fotoaparati in GPS naprave.

Mobilne naprave nam omogočajo uporabo tako samostojnih mobilnih aplikacij kot spletnih strani, zaradi česar se podjetja in posamezni razvijalci srečujejo s dilemo, ali za uporabnike mobilnih naprav izdelajo produkt kot samostojno mobilno aplikacijo ali pa omogočijo uporabo takšnega produkta preko spletne strani. Vsaka od opcij ima svoje prednosti in slabosti, ki se kažejo tako v ceni kot funkcionalnostih. Pomembna razlika med spletno stranjo in samostojno aplikacijo je predvsem dejstvo, da je spletno stran možno videti tako na osebnem računalniku, kot različnih platformah mobilnih napravah, medtem ko so samostojne mobilne aplikacije omejene na mobilne naprave s točno določeno platformo (npr. samo za android).

Spletne strani se uporabniku prikazujejo s prenosom podatkov preko interneta v obliki povezanih HTML strani, dodatno pa so vizualno urejene s jezikom CSS. Funkcionalnost spletih strani in obdelavo podatkov omogočamo s pomočjo programskih jezikov kot je Javascript, PHP, Java, C#. Spletne strani so shranjene na strežniku in tečejo znotraj brskalnikov, kar pomeni da so na voljo vsem uporabnikom, ki imajo na voljo internet in brskalnik, razen v primeru, ko strežnik ali internetna povezava ne delujeta.

Podobno kot pri spletnih straneh, imamo tudi pri mobilnih aplikacijah vizualni del za interakcijo in v ozadju programsko kodo, ki omogoča funkcionalnost in obdelavo nekih podatkov. Interakcija se pri aplikaciji izvaja na sami mobilni napravi, kjer je ta aplikacija naložena, medtem ko se obdelava podatkov lahko izvaja lokalno na napravi ali pa se za obdelavo in hranjenje prenaša na strežnik. Če želimo uporabljati mobilne aplikacije, jih je potrebno predhodno naložiti na mobilno napravo, kar pa pri spletnih straneh ni potrebno. Ker mobilne aplikacije obstajajo izključno na posamezni mobilni napravi, se tudi izvajajo samo za

uporabnika te naprave. Programski jeziki, za izdelavo aplikacije, so odvisni od platforme mobilne naprave, posledično je potrebno za vsako platformo izdelati novo različico aplikacije.

Prednosti mobile aplikacije:

- Boljše pri bolj interaktivnih aplikacijah, kjer uporabnik uporablja veliko različnih funkcij (npr. igre).
- Možne kompleksnejše kalkulacije.
- Možne so osebne prilagoditve aplikacije kot je izgled in delovanje funkcikonalnosti, kar je pri spletnih aplikacijah možno samo v omejenem obsegu.
- Možna uporaba vseh funkcionalnostih mobilnih naprav (SMS, kamera).
- Ni potrebna internetna povezava, saj so vsi podatki na sami napravi in prenos ni potreben.

Prednosti spletnih strani:

- Takojšna dostopnost, brez predhodne naložitve programa na sistem.
- Spletne strani so kompatibilne s vsemi napravami, kar omogoča tudi večjo dostopnost vsem uporabnikom.
- Možna takojšna nadgradnja ali sprememba spletne strani za vse uporabnike hkrati.
- Lažje najti na spletu zaradi iskalnikov.
- Lažje izdelati in posledično cenejši produkt.

Za diplomsko delo, smo se odločili izdelati aplikacijo, ki nam s pomočjo zemljevidov sistema Google Maps, omogoča vizualni prikaz trenutnega geografskega položaja oddaljene mobilne naprave. Aplikacijo smo izdelali v dveh različicah, in sicer samostojno mobilno aplikacijo za platformo Android ter spletno različico iste aplikacije. Za prikaz in delovanje zemljevida smo uporabili sistem Google Maps, ki nam omogoča prikaz zemljevida s ptičje perspektive, za določanje geografskega položaja mobilne naprave pa smo uporabili sistem GPS, ki omogoča pridobivanje koordinat s pomočjo satelitov. Orodje, ki smo ga uporabili za izdelavo mobilne aplikacije za platformo Android, je bilo orodje Android Studio, medtem ko smo spletno

različico iste aplikacije izdelali s pomočjo programa Notepad (slovensko: Beležka) ter spletnim brskalnikom Firefox. Obe različici smo nato primerjali in določili razliko v kompleksnosti in trajnosti izdelave ter uspešnosti delovanja same aplikacije. Aplikacijo smo testirali na mobilni napravi LG e975.

2 Tehnologije in orodja

2.1 Mobilne naprave

Mobilna naprava [8], je vsaka naprava manjše velikosti, ki ima računalniške zmogljivosti. Dejansko je to majhen računalnik., ki ga držimo v roki. Mobilne naprave imajo svoj operacijski sistem, ki vsebuje svoje aplikacije. Najpogosteje imajo mobilne naprave zmogljivost predvajanja zvoka, videa, povezave na internet preko Wi-Fi in druge zmogljivosti, ki jih ponavadi najdemo pri standardnih računalnikih. Najbolj tipični primeri mobilnih naprav so tablični računalniki in telefoni pri katerih vnašamo podatke preko miniaturne tipkovnice ali zaslona na dotik.

2.2 Tablični računalniki

Tablični računalniki [9] so vmesni člen med telefonom in osebnim računalnikom. Vnos podatkov poteka preko dotika na zaslon, Grafični vmesni ponuja navidezno tipkovnico po kateri lahko tipkamo s prstom, objekte klikamo in vlečemo/premikamo s prstom na ekranu. Sama naprava vsebuje kamero, senzorje, ki zaznavajo premikanje naprave, zvočnike in par gumbov za osnovno upravljanje naprave, kot je sprememba zvoka. Tablice so ponavadi večje od telefonov, in sicer diagonalno merijo od 18 centimetrov navzgor.

2.3 Pametni telefoni

Pametni telefoni [10] so mobilne naprave, podobne tabličnim računalnikom. Delujejo na posebnih mobilnih operacijskih sistemih, ki so podobni računalniškim, z nekaj dodatnimi funkcionalnostmi. Tako kot tablični računalniki, imajo moderni pametni telefoni virtualno tipkovnico (starejši uporabljajo fizično miniaturno tipkovnico), senzorje, kamero, zvok, aplikacije in ostale stvari. Najbolj opazna razlika med tabličnim računalnikom in pametnim telefonom je razlika v velikosti, saj so pametni telefoni dovolj majhni, da jih lahko držimo v eni roki.

Zaradi velikega števila uporabnikov pametnih telefonov in posledično zelo različnih uporabniških potreb, je na voljo tudi zelo veliko različnih aplikacij. Takšno raznolikost

omogočajo spletne trgovine, kjer lahko uporabniki enostavno najdejo in naložijo raznovrstne aplikacije. Te aplikacije so ponavadi zelo poceni ali kar brezplačne.

2.4 GPS

Globalni sistem pozicioniranja [1] (ang.: *Global Positioning System* ali *GPS*), je satelitski navigacijski sistem, ki se uporablja za določanje točne lege in časa kjerkoli na Zemlji. Zasnovali so ga na obrambnem ministrstvu v ZDA, dostopen pa je javno vsakemu, ki ima ustrezen sprejemnik. Sistem je sestavljen iz minimalno 24 satelitov, ki se nahajajo v 6 ravninah tirnic. Vsak od njih obkroži Zemljo dvakrat dnevno na višini več kot 20000 km. Sateliti neprestano oddajajo čas (imajo atomsko uro) in podatke o tirnici gibanja, ki jih določajo opazovalnice na Zemlji.

Lokacijo nekega sprejemnika na Zemlji določimo s uporabo 4 satelitov hkrati. Sprva določimo razdaljo med sprejemnikom in satelitom, ki jo dobimo kot razliko med časom sprejema signala in časom njegove oddaje. Nato iz njihovih signalov in notranje baze podatkov ugotovimo mesta satelitov. Sprejemnik se torej nahaja na sferi, katere središče je satelit in katere polmer je določen z razdaljo, ki jo premagajo radijski signali v času od trenutka oddaje do trenutka sprejema signala. Ker sprejemnik hkrati sprejema signale iz več satelitov, je mogoče določiti lego sprejemnika na osnovi presečišča sfer s posameznih satelitov.

2.5 Internet

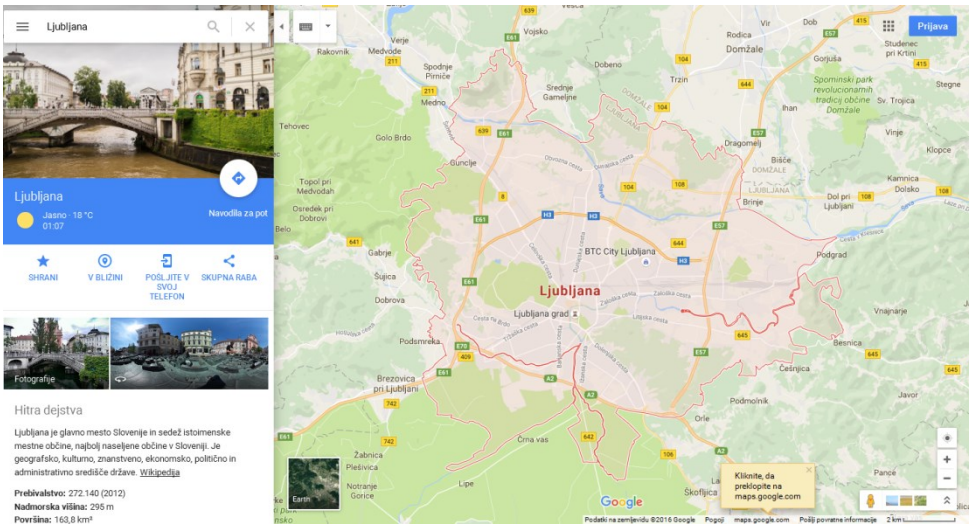
Internet [2] je globalni sistem medsebojno povezanih računalniških omrežij, ki preko protokolov TCP/IP omogočajo komunikacijo med milijardami naprav na svetu. Internet nam omogoča prenos informacij in storitev kot so dokumenti s nadbesedilom (ang.: hypertext), aplikacije, eletronske pošte, telefonske storitve in drugo.

2.5.1 Model odjemalec - strežnik

Komunikacija med 2 napravama ponavadi poteka preko interneta po modelu odjemalec-strežnik [3], pri čemer odjemalec pošilja zahteve na nek strežnik in prejema odgovore tega strežnika. Odjemalce in strežnike klasificiramo na podlagi tega, kaj nam ponujajo. Tako imamo spletne strežnike in spletne odjemalce za delo s spletnimi stranmi, datotečne strežnike in odjemalce za delo s datotekami in podobno. Pri sistemu spletišč imamo torej odjemalca (brskalnik), s katerim pošiljamo zahteve in strežnik, ki odgovarja odjemalcu (na podlagi zahtev) v obliki spletnih strani ter njihovih vsebin.

2.6 Google Maps

Je popularen in dostopen sistem na spletnem strežniku, ki omogoča geografski prikaz neke lokacije na zemljevidu. Med drugim je možen ogled zemljevida sveta, posameznih držav, mest. Za naše potrebe nam sistem Google Maps [4] omogoča tudi prikaz geografskega položaja na podlagi koordinat. Zemljevidi so izrisani kot kombinacija slik izdelanih s letali in satelitskih slik, ki jih pri podjetju Google dokaj redno posodabljajo. Google Maps (Slika 2.1) deluje s pomočjo programskih in označevalnih jezikov kot je programski jezik Javascript, XML, Ajax itd.



Slika 2.1: Zemljevid sistema Google Maps.

2.6.1 Google Maps API

Google Maps API [4] je brazplačen API, ki omogoča uporabnikom implementacijo zemljevidov in ostalih storitev sistema Google Maps, na svoje spletne strani in aplikacije. Ker je sistem Google maps brezplačen tudi za komercialno uporabo in ga je enostavno implementirati na spletne strani, je sistem zelo priljubljen med razvijalci spletnih strani in programske opreme.

2.7 Spletne tehnologije

Spletne tehnologije so različne tehnologije, ki nam omogočajo izdelavo in delovanje spletnih strani in z njimi povezanimi dejavnostmi. Med takšne tehnologije spadajo različni programski

jeziki (npr.: java, c#, php), označevalni jeziki (npr.: html), podatkovne baze (npr.: mysql), internet in podobno.

2.7.1 Spletišče

Spletišče [5] (ang.: website) je skupek spletnih strani (ang.: web pages), ki so vrsta dokumentov s nadbesedilom (ang.: hipertekst). Povezave med posameznimi spletnimi stranmi nam omogočajo hiperpovezave (ang.: hyperlinks), do spletišč pa dostopamo preko internetnega protokola (ang.: internet protocol ali IP) oziroma človeku prijazne oblike, ki ji pravimo domena. Spletne strani prikazujemo v posebnih programih imenovanih spletni brskalniki (ang.: browser) kot je naprimer Firefox ali Chrome. Zgradbo spletne strani opisujemo s opisnim jezikom HTML [6], oblikovanje, barve, vrsto pisave pa določimo s označevalnim jezikom CSS.

V kolikor želimo imeti spletno stran za uporabnika bolj interaktivno, jo lahko izpopolnimo s skriptnim jezikom javascript ali njegovo popularno knjižico jQuery, ki programerju omogoča enostavn dostop in manipulacijo elementov HTML in njihove vsebine.

Ker pri spletnih straneh pogosto delamo s veliko podatkov, ki jih želimo v ozadju hraniti in obdelovati za kasnejšo uporabo, nam to možnost omogočajo podatkovne baze (mysql) ter programski jeziki kot je php, c#, java in podobni.

2.8 Operacijski sistem Android

Android [11] je operacijski sistem, izdelan na podlagi operacijskega sistema Linux, za mobilne naprave kot so tablični računalniki in pametni telefoni. Tako kot Linux, je Android odprtokodni sistem in posledično zelo priljubljen pri neodvisnih razvijalcih in podjetjih, ki hočejo pripravljen odprotokodni operacijskih sistem. V letu 2015 je bil Android, s svojim logotipom zelenega robotka (Slika 2.2), najbolj razširjen mobilni operacijski sistem na svetu. Sledi mu iOS od podjetja Apple.

Slika 2.2: Logotip operacijskega sistema Android.

Android deluje na podlagi neposredne manipulacije objektov preko zaslona na dotik. Polega mobilnih naprav so pri podjetju Google razvili tudi različice Android za avtomobile, ure in televizorje. Android deluje na arhitekturi ARM (Advanced RISC Machine), pri novejših različicah pa uporablja tudi procesorje Intel. Novejše različice so možne tako z 32 kot 64 bitnim sistemom.

2.8.1 Vmesnik operacijskega sistema Android

Z objekti lahko neposredno manipuliramo s premiki prstov po zaslonu. Možno je pritiskati gumbe, premikati objekte, jih povečati ali zmanjšati z preprostim ščipanjem po ekranu. Podatke lahko vnašamo preko virtualne tipkovnice, možno pa je tudi priklopiti zunanje naprave (naprimer tipkovnico), s katerimi lahko manipuliramo dogajanje znotraj telefona.

Ker ima sama naprava s operacijskim sistemom Android [16] vgrajene senzorje za pospeševanje, premikanje in ostale stvari, imajo aplikacije na tem sistemu možnost vidne odzivnosti v smislu vibriranja, spreminjanja navpičnega pogleda v vodoravni pogled, podajanja drugih informacij.

Na prvi (domači) strani naprave s sistemom Android, se na zaslonu nahaja navigacija, ikone in programi, ki dajejo podobno delovanje kot namizje na osebnih računalnikih. Ikone dajejo možnost zagona aplikacij, medtem ko posamezni programi podajajo podatke *v živo* (tipičen primer takega programa je ura). Domača stran je razdeljena na več podstrani, med katerimi lahko menjamo z povleko prsta po zaslonu. Zgoraj ima Android informacijsko vrstico, ki nam podaja trenutno stanje. Med drugim najdemo tam informacije o spregledanih sms-ih, telefonskih klicih in nadgradnjah aplikacij.

2.8.2 Aplikacije operacijskega sistema Android

Aplikacije za android [18] programiramo s pomočjo programskega jezika Java [13] in označevalnega jezika XML [14]. Označevalni jezik XML omogoča grafično oblikovanje in postavitve elementov, ki so del aplikacije, programski jezik Java pa nam omogoča izvajanje funkcionalnosti aplikacije.

- XML(Extensible Markup Language) [17] je označevalni jezik, podoben jeziku HTML. Je jezik, s katerim formalno opisujemo neko strukturo podatkov. Berljiv je tako za človeka kot naprave. Na spletu ga uporabljamo predvsem za prenos podatkov v strukturirani obliki. Seveda pa ga je možno brati tako s spletnih brskalnikov kot posameznih aplikacij.

  Strukture jezika XML so sestavljene iz etikete, elementa in atributa:

  - Element je nek podatek, ki ga pošiljamo znotraj strukture (npr. ime, količina).
  - S etiketo določimo začetek in konec nekega elementa (npr.: *<etiketa>element</etiketa>*).
  - S atributom določimo dodatne lastnosti elementa (npr.: spol).

- Java [16] je objektno usmerjen programski jezik, ki so ga razvili pri podjetju Sun Microsystems. Je eden najbolj razširjenih programskih jezikov na svetu. Je neodvisen od platforme in prirejen za splošno uporabo. Njegova koda je napisana znotraj razredov, podatki pa so predstavljeni kot objekti (izjeme so primitivni podatkovni tipi kot je integer, boolean itd.).

  Primer programa, napisanega v programskem jeziku Java, s imenom Primer:

```
class Primer {
     public static void main(String[] args){
     System.out.println("Zdravo!"); // v konzolo izpiše
Zdravo.
     }
}
```

Lastnosti aplikacij na sistemu Android:

- Operacijski sistem Android deluje na podlagi večuporabniškega sistema Linux, zato se vsaka aplikacija poganja kot ločen uporabnik.

- Linux določi vsaki aplikaciji svoj ID (identifikator) in temu ID določi pravice za uporabo datotek te aplikacije.
- Vsaka aplikacija teče ločeno od drugih aplikacij, znotraj posamezne virtualne naprave.
- Vsaka aplikacija je za sistem Linux ločen proces.

Tak sistem delovanja omogoča izvajanje aplikacij po principu najnižjega privilegija (angleško *principle of least privilege*). To omogoča večjo varnost, saj aplikacije dostopajo zgolj do komponent, ki jih potrebujejo za delovanje. Za višje privilegije, kot je naprimer dostop do SMS, fotoaparat in kamero, mora uporabnik odobriti delovanje aplikacije. Pi želji programerja je možno tudi urediti aplikacije, da se izvajajo pod istim ID in istim procesom, znotraj iste virtualne naprave, z namenom prihraniti na porabi sistemskih virov.

Sestavni deli operacijskega sistem Android za delovanje aplikacij [12]:

- Aplikacije – programi, ki delujejo kot posamezni procesi znotraj Linux.
- Aplikacijsko ogrodje, združuje vse sistemske aplikacije.
- Prevajalnik – prevede kodo v napravi razumljivo obliko.
- Knjižice – del sistema, ki razvijacem omogoča upravljanje s strojnimi komponentami.

Komponente androidne aplikacije

Komponenta [7] je posamezen sestavni del neke večje androidne aplikacije, pri čemer ima vsaka komponenta svoj cilj in delovanje. Komponente lahko delujejo samostojno ali pa so medsebojno povezane. Lahko so enostavne ali pa sestavljene iz večjih manjših delov oziroma funkcij.

Poznamo štiri vrste komponent, od katerih ima vsaka svojo vlogo pri delovanju aplikacije:

- Aktivnosti (angleško activities) – Vsaka aplikacija deluje sestavljeno iz različnih aktivnosti, ki so predstavljene kot posamezni zasloni, vsak s svojim uporabniškim vmesnikom. Tako je lahko aktivnost zaslon, ki nam prikazuje listo nove prejete elektronske pošte, medtem ko druga aktivnost omogoča sestavo novega sporočila in tretja omogoča branje sporočil. Ker aktivnosti delujejo neodvisno ena od druge, je

možno, da aktivnost ene aplikacije, zažene aktivnost druge aplikacije, vendar samo, če ji ta druga aplikacija to dovoli. Aktivnost je lahko predstavljena tudi transparentno kot dialogno okno (naprimer izbira datuma). Vsaka aktivnost je podrazred razreda *Activity*.

- Storitve (angleško services) - so komponente, ki tečejo v ozadju in opravljajo dolgotrajne operacije ali pa opravljajo delo za oddaljene procese. Primer take operacije je predvajanje glasbe v ozadju ali prenos podatkov, medtem ko uporabnik brska po internetu. Storitve so podrazred razreda *Service*.

- Ponudniki vsebine (angleško content providers) – so upravljalci za skupno rabo podatkov. Preko teh ponudnikov, lahko različne aplikacije dostopajo do podatkov in jih tudi spreminjajo. Tak primer je ponudnik vsebine znotraj sistema Android, ki omogoča dostop do podatkov o kontaktih. Podatki so lahko shranjeni v SQLite podatkovni bazi, sistemskih datotekah, na spletu ali pa kjerkoli drugje, kjer lahko aplikacija dostopa do podatkov. Ponudnik vsebine je podrazred od razreda *ContentProvider*.

- Sprejemniki obvestil (*broadcast receiver*) – so komponente, ki regirajo na obvestila celotnega sistema. Primer takšne komponente je obveščanje, da je baterija prazna. Takšne komponente so ponavadi del sistema, lahko pa se obveščanje izvede tudi s strani posameznih aplikacij, ki sporočajo drugim aplikacijam o nekem dogodku (naprimer obvestilo, da je bil končan prenos podatkov in so ti zdaj na voljo za uporabo). Sprejemniki obvestil nimajo uporabniškega vmesnika, lahko pa se znotraj »status bar« ustvari opozorilo za uporabnika. Sprejemnik obvestil je podrazred razreda *BroadcastReceiver*.

Posebnost sistema Android je, da lahko aplikacije kadarkoli aktivirajo komponente drugih aplikacij. Če želimo ustvariti aplikacijo, ki vključuje zajemanje slik s fotoaparatom in za takšno dejanje že obstaja aplikacija, ki ima takšno funkcijo, lahko s svojo aplikacijo dostopamo do komponente te druge aplikacije, ki opravlja zajemanje slik. Do komponente dostopamo tako, da zaženemo aktivnost v aplikaciji, katere komponeto želimo uporabiti. Aplikacije ne zaženejo komponent drugih aplikacij neposredno ampak preko sistema Android, ki prejme prošnjo za zagon komponente druge aplikacije, preveri namen (angleško intent) za zagon neke komponte ter jo nato tudi zažene.

Vsaka aplikacija teče kot ločen proces, ki se zažene ob uporabi neke komponente te aplikacije. Tako teče komponenta aplikacije za zajemanje slik v ločenem procesu od naše aplikacije, ki uporablja to komponento.

Aktivacija komponent

Vse komponente, razen »ponudniki vsebine«, različnih aplikacij se med izvajanjem medsebojno povezujejo na podlagi asinhronskega obveščanja imenovanega »namen« (ang.: intent). Nameni so kreirani kot objekti tipa Intent, z njimi pa definiramo obvestila, s katerimi aktiviramo točno določene komponente (explicitno) ali pa aktiviramo neko vrsto oziroma tip komponent(implicitno).

Pri aktivnostih in storitvah, namen določi, kaj naj se izvede. Pri tem lahko poda informacije o naslovu do podatkov, ki jih potrebuje za delo. Primer takšne izvedbe je zagon spletnega brskalnika ali pa prikaza neke slike. Za razliko od aktivnosti in storitev pa se pri sprejemnikih obvestil zgolj definira katero obestilo se sprejme.

Za razliko od prej omenjenih komponent pa je komponenta ponudnik vsebine (angleško content provider) aktivirana na podlagi zahteve od razreda ContentResolver.

Metode za aktivacijo posameznih komponent zaganjamo s uporabo Intent skupaj s metodami:

- Aktivnosti zaženemo s metodo startActivity() (ne vača rezultat) ali startActivityForResult() (vrača rezultat).
- Storitve zaženemo s metodami startService() ali bindService().
- Sprejemnike obvestil poganjamo s metodami sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
- Poizvedbe po podatkih lahko izvajamo s pomočjo metode query(), ki jo najdemo v razredu ContentResolver.

Datoteka MANIFEST

Vsaka aplikacija mora imeti datotetko po imenu AndroidManifest.xml, v kateri je obvezno deklarirati, katere komponente bo naša aplikacija uporabljala.

V tej datoteki torej določimo naslednje podatke:

- Deklariramo vse komponente, ki jih uporablja.
- Določimo uporabniška dovljenja (naprimer dovoljenje za dostop do interneta).
- Določimo, katera je najstarejša različica API, ki jo je možno uporabljati.
- Določimo, katere API knjižice bo naša aplikacija uporabljala.
- Deklariramo podatke o strojni ali programski opremi, ki jo aplikacija uporablja.

Primer deklaracije za komponento za neko aktivnost:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity
android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

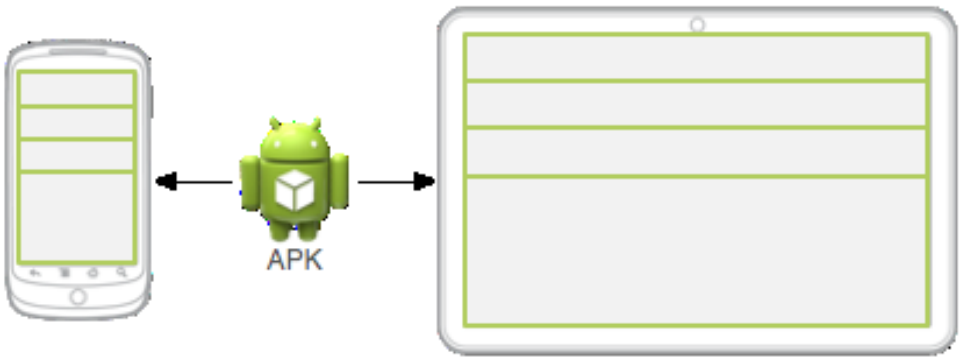
Kot je vidno v zgornjem primeru, pišemo podatke v datoteko AndroidManifest.xml, v jeziku XML, pri čemer deklariramo aktivnosti znotraj značk <activity>, storitve znotraj <services>, ponudnike vsebine znotraj <providers> in sprejemnike obvestil znotraj <recievers>. Vse komponente so zavite v značke <application>, pri kateri imamo določen atribut »android-icon«, ki preko spletnega naslova URL, dostopa do ikone aplikacije.

Viri podatkov in slik

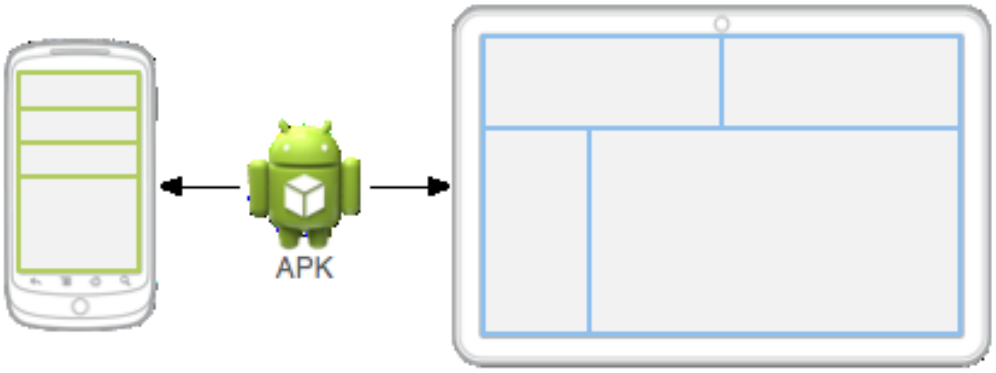
Ker imamo na razpolago veliko različnih naprav s različnimi konfiguracijami in različnimi velikostmi slik, je priporočeno hraniti slike in nize (angleško strings) ločno od ostale kode aplikacije. Prav tako nam to olajša vzdrževanje posameznih delov aplikacije. Zaradi organizacije hranimo slike in nize znotraj mape »res/«, ki je razdeljena v podmape z različnimi različicami sistema.

Viri [20] so razdeljeni v dve skupini:

- Osnovni (angleško default) viri so viri, ki jih vsaka naprava uporabi, če nima na voljo alternativih virov, ki se ujemajo z napravo (Slika 2.3).
- Alternativni viri, so viri prilagojeni za posamezne naprave (Slika 2.4).



Slika 2.3: Dve napravi, ki uporabljata iste osnovne vire.



Slika 2.4: Dve napravi, ki uporabljata različne (alternativne) vire.

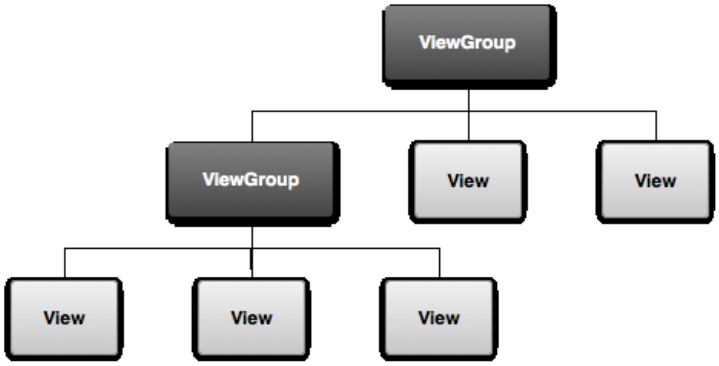
V primeru, da imamo za neko napravo dve možni različici slik ozadja (glede na obrnjenost naprave), shranimo osnovno različico slike v mapo *res/layout/,* alternativno različico pa v mapo *res/layout-land/.* Android izbere primerno sliko samodejno.

Pogledi (View) in Skupine pogledov (ViewGroup)

Uporabniški vmesniki sistema Android [21] so zgrajeni na podlagi dveh vrst objektov:

- Objekt tipa View (slovensko pogled) – na zaslon izrisuje stvari, s katerimi imajo uporabniki interakacijo.
- Objekt tipa ViewGroup (slovensko skupina pogledov)– skupina pogledov (View), ki definira izgled vmesnika.

Objekti tipa *View* in objekti tipa *ViewGroup* skupaj sestavljajo drevesa (Slika 2.5), pri čemer so *ViewGroup* nevidna ogrodja, ki združujejo več različnih objektov tipa *View*.



Slika 2.5: Drevo pogledov (*View*) in skupin pogledov (*ViewGroup*).

Iz razredov *View* in *ViewGroup* so izpeljani podrazredi, s katerimi definiramo gumbe, vnosna polja ter različne vrste postavitve (npr. RelativeLayout in LinearLayout).

Prej omenjene postavitve, vnosna polja ter gumbe, lahko definiramo kot objekte v jeziku Java, lahko pa jih tudi opišemo v jeziku XML. Pri tem določimo opisno polje v jeziku Java kot objekt tipa *TextView* ali pa alternativno v jeziku XML s označbo *TextView*. Sistem Android nato inicializira vse sestavne dele, ki so opisani v XML in jih lahko uporabljamo kot del kode jezika Java.

Primer definicije gumbov in opisnih polj:

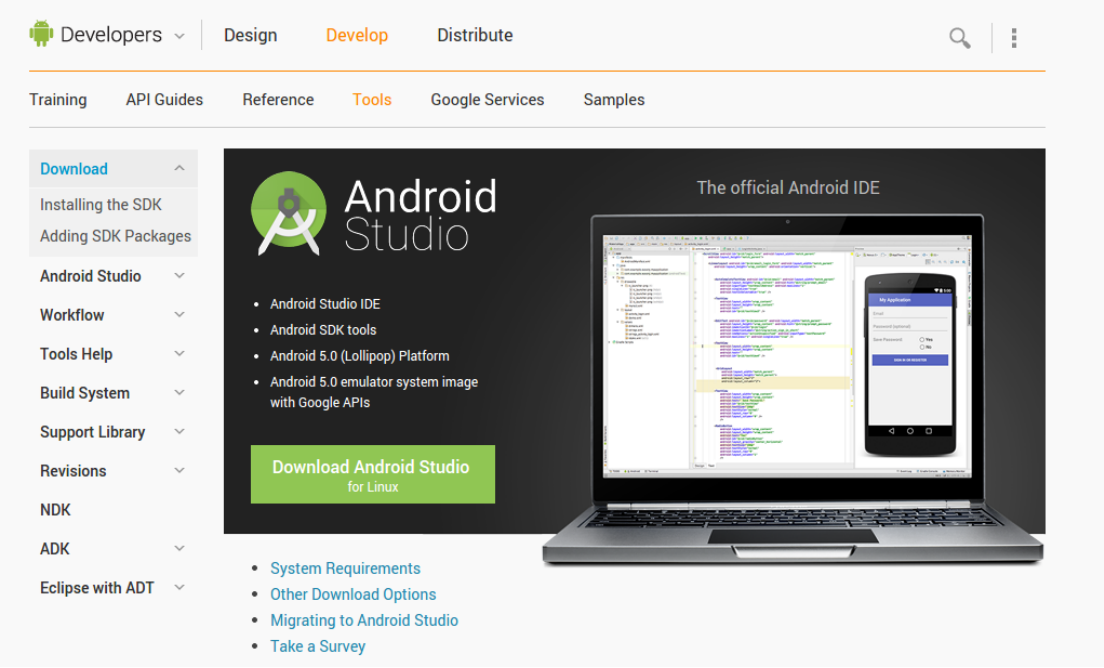
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="I am a Button" />
</LinearLayout>
```

2.9 Razvojno okolje Android Studio

Android Studio [15] je uradno razvojno okolje, narejeno s strani podjetja Google in je trenutno najbolj priljubljeno razvojno okolje za programiranje aplikacij za operacijski sistem Android. Vsebuje inteligenten urejevalnik, ki samodejno dokončuje in analizira kodo, kar omogoča hitrejše in enostavnejše pisanje programske kode. Prednost uporabe tega razvojnega okolja je tudi hitro poganjanje aplikacij tako znotraj emulatorja kot neposredno na mobilni napravi. Android Studio nam tudi omogoča uporabo predlog za različne mobilne naprave in vnaprej generirano programsko kodo za nekatere aplikacije ali posamezne funkcionalnosti, ki jih lahko uporabljamo. Omogoča nam tudi enostavno preklapljanje med programiranjem v jeziku Java in programiranjem uporabniškega vmesnika v jeziku XML, kar nam še poveča enostavnost in hitrost izdelave aplikacij.

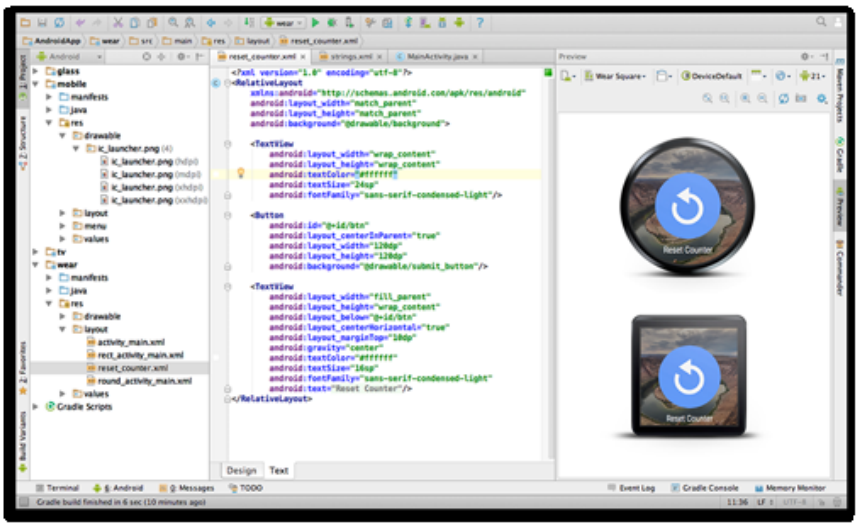
Razvojno okolje Android Studio [19] lahko prenesemo na naš računalnik s uradne strani http://developer.android.com/sdk/index.html (Slika 2.6).



Slika 2.6: Uradna stran za namestitev okolja Android Studio.

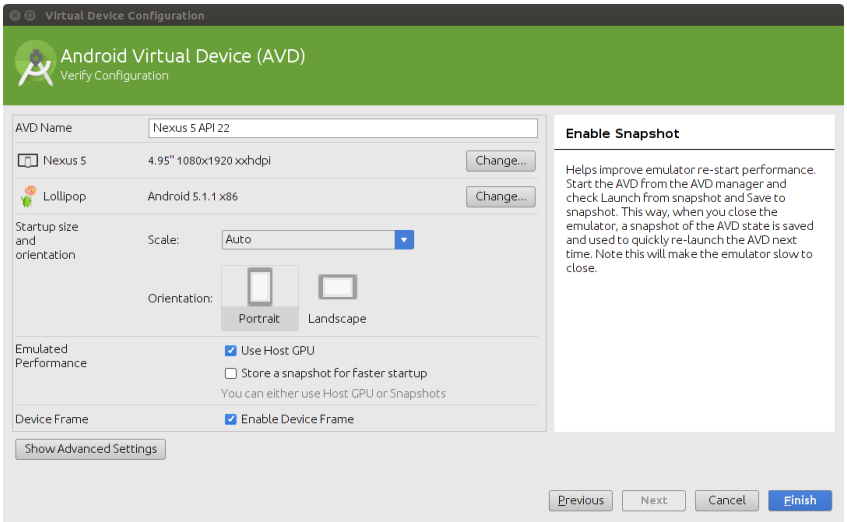
2.9.1 Razvoj aplikacij v Android Studio

Razvoj aplikacij poteka v večokenskem načinu (Slika 2.7). Tipično je na levi strani drevo map, slik in datotek, ki sestavljajo aplikacijo. Na sredini je omogočen način pisanja programske kode v jezikih XML in Java. Na desni pa je predogled grafičnega vmesnika, za katerega lahko izbiramo med predogledi za različne naprave s sistemom Android.



Slika 2.7: Razvojno okolje Android Studio v večokenskem načinu.

Android Studio vsebuje tudi emulator. S tem emulatorjem je omogočena uporaba Android Vitual Device (kratica: AVD) (Slika 2.8), ki omogoča predogled, kako bo aplikacija izgledala in delovala na neki napravi Android. AVD omogoča, da testiramo aplikacijo na različnih različicah in velikostih naprav v realnem času.



Slika 2.8: Nastavitve za zagon Android Virtual Machine (AVD).

3 Razvoj mobilne aplikacije Sledilec

3.1 Ideja in namen

V času mobilnih naprav se podjetja vse pogosteje srečujejo z dilemo, ali za uporabnike mobilnih naprav izdelajo produkt kot mobilno aplikacijo ali pa omogočijo uporabo produkta preko spletne strani. Vsaka od opcij ima svoje prednosti in slabosti, ki se kažejo tako v ceni kot funkcionalnostih. Pomembna razlika med spletno stranjo in aplikacijo je predvsem tudi dejstvo, da je spletno stran možno videti tako na osebnem računalniku, kot različnih platformah mobilnih napravah, medtem ko so mobilne aplikacije omejene na mobilne naprave s točno določeno platformo (npr. operacijski sistem Android).

Tako smo se za to diplomsko delo odločili izdelati dve različici aplikacije, ki nam omogoča prikaz geografskega položaja oddaljene mobilne naprave. Prvo različico aplikacije smo izdelali izključno za mobilno platformo Android, medtem ko smo drugo različico izdelali kot navadno spletno storitev, ki jo lahko poganjamo preko različnih naprav, neglede na platformo.

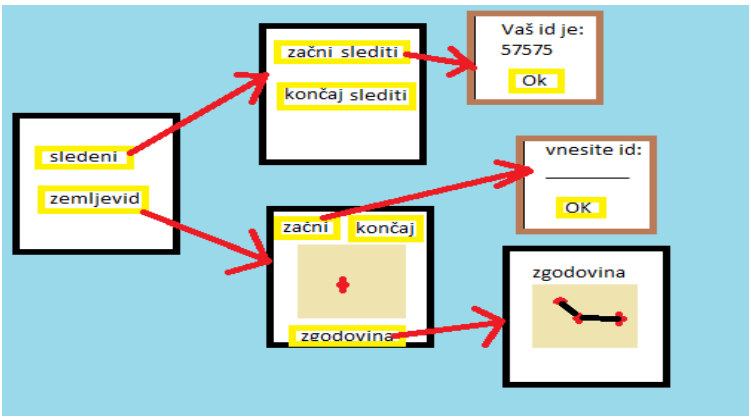
Aplikacija omogoča uporabniku, da s storitvijo Google Maps, na zemljevidu vidi trenutno lokacijo oddaljene mobilne naprave.

3.2 Načrtovanje

Prva naloga je bila izdelava osnovne skice vseh sestavnih delov, ki bi jih takšna aplikacija naj vsebovala ter zgodbe (ang.: storyboard), ki opisuje delovanje takšne aplikacije (Slika 3.1). Aplikacija bi za svoje delovanje imela strežnik in odjemalce. Odjemalec bi uporabniku omogočal vizualni prikaz lokacije oddaljene mobilne naprave na zemljevidu ali pa pošiljanje svojih koordinat na strežnik. Strežnik bi prejemal in hranil podatke o koordinatah posameznih mobilnih naprav ter jih posredoval drugemu odjemalcu, ki bi od njega zahteva koordinate. Omejitev, kdo sme prejemati določene koordinate, bi omogočili s pomočjo identifikacijske številke, ki jo strežnik avtomatsko generira za mobilno napravo, ki pošilja koordinate. S to številko lahko potem uporabniki zahtevajo in prejemajo te koordinate od strežnika.

3.2.1 Zahteve

- Tako oddajnik kot prejemnik koordinat, imata enak grafični vmesnik s enakimi opcijami in lahko sledita en drugemu.
- Ob zagonu aplikacije ima uporabnik možnost zagona oddajanja koordinat (oddajnik) ali pa zagona sledenja oddaljeni mobilni napravi (prejemnik).
- V kolikor uporabnik izbere opcijo za sledenje oddaljeni mobilni napravi, se mu prikaže zemljevid, na katerem lahko ob pritisku na gumb *začni slediti* vpiše identifikacijsko številko in nato vidi lokacijo te mobilne naprave..
- Uporabnik lahko izbere tudi opcije za prikaz in izklop zgodovine sledenja ter zaustavitev trenutnega sledenja.
- V kolikor uporabnik izbere opcijo za oddajanja koordinat, se mu prikažejo gumbi za zagon in zaključek oddajanja koordinat. Ob pritisku na gumb za zacetek oddajanja koordinat, se uporabniku izpiše identifikacijska številka, s katero lahko omogoči drugim uporabnikom, da mu sledijo.



Slika 3.1: Zgodba (Storyboard) aplikacije.

3.3 Implementacija

3.3.1 Strežnik

Predno smo se lotili izdelave aplikacije, smo postavili strežnik in podatkovno bazo. Ustvarili smo dve datoteki, ki s programskim jezikom PHP sprejemata podatke in vračata odgovor.

Datoteka za prejem podatkov (*prejem.php*)

Podatke, ki smo jih pošiljali na strežnik, smo obdelali s pomočjo programskega jezika PHP, s katerim preverimo podatke in jih zapišemo v podatkovno bazo strežnika.

Takšna datoteka vključuje:

- Povezavo na podatkovno bazo.
- Prejem podatkov preko spremenljivke *POST:*

```
$idUser=$_POST['idUporabnika'];
$dolzina=(string)$_POST['latitude'];
$visina=(string)$_POST['longitude'];
```

- Preverjanje podatkov in zapis teh podatkov v podatkovno bazo:

```
mysql_query("INSERT INTO imeTabele
VALUES(NULL,'$odgovor,'$dolzina,'$visina) ",$con);
```

- Vrnitev odgovora v obliki *JSON:*

```
print json_encode($odgovor);
```

Datoteka za oddajanje podatkov (*koordinate.php*)

S to datoteko smo omogočili, da uporabniki pridobivajo podatke o lokaciji drugih mobilnih naprav.

Datoteka vsebuje podobno sestavo kot datoteka *prejem.php*:

- Povezava s podatkovno bazo.
- Prejem podatkov preko spremenljivke *POST (*prejmemo uporabniški ID).
- Obdelava podatkov, kjer (za razliko od datoteke *prejem.php)*, preverimo, če uporabniški ID obstaja v podatkovni bazi in če lahko iz baze preberemo ter obdelamo podatke o lokaciji tega uporabniškega ID.
- Sledi branje podatkov iz podatkovne baze, in sicer vrstice kjer se identifikacijska številka ujema s atributom *'ime':*

```
$sql = "SELECT dolzina, sirina FROM imeTabele WHERE ime LIKE
$idUporabnika";
$odgovor = $con->query($sql);
if ($odgovor->num_rows > 0) {
   while($row = $result->fetch_assoc()) {
      $lat=$row["dolzina"]; //preberemo dolžino
      $lon=$row["sirina"]; //preberemo višino
   }
}
```

- Zadnja vrstica programske kode je odgovor s strani strežnika v obliki JSON, ki nam pošlje koordinate:

```
print json_encode($lat.";".$lon);
```

3.3.2 Aplikacija - *različica* za Android

Takoj ob zagonu aplikacije se uporabniku prikaže prva stran s dvemi opcijami (Slika 3.2), ki nas preusmerita na željeno aktivnost:

- opcija za oddajanje koordinat (*oddajnik*),
- opcija za prikaz lokacije oddaljene mobilne naprave na zemljevidu (*prejemnik*).

Slika 3.2: Prva stran aplikacije za sledenje uporabniku (različica za Android).

Oddajnik

Oddajnik je del aplikacije, ki omogoča oddajanje podatkov o geografskemu položaju mobilne naprave. Uporabnik ima za uporabo na voljo opcije:

- Začni oddajati koordinate.
- Zaključi oddajanje koordinat.

Grafični vmesnik

Vsak vizualni prikaz elementov na zaslonu, vidimo kot novo *aktivnost*. Prikazujemo jo kot strukturo različnih gradnikov v jeziku XML. Pri tem se na najvišjem nivoju nahaja gradnik, ki določa postavitev elementov znotraj sebe (npr.: *LinearLayout*). Ta postavitev je lahko vodoravna, navpična ali pa relativna glede na druge elemente. Znotraj tega gradnika lahko postavljamo raznorazne gradnike kot so vnosna polja, gumbi, dialogna okna, gradniki za prikaz spletne strani in ostalo.

Nekateri gradniki, ki smo jih uporabljali za *oddajnik* (Slika 3.3):

- ***LinearLayout*** – gradnik za urejeno postavitev gradnikov navpično ali vodoravno.
- ***RelativeLayout*** – gradnik za relativno pozicioniranje gradnikov glede na druge gradnike znotraj RelativeLayout.
- ***TextView*** - prikaz besedila.
- ***Button*** – gradnik za gumb.

Slika 3.3: Grafični vmesnik za oddajnik (različica za Android).

Strukturo grafičnega vmesnika za oddajnik smo torej sestavili s pomočjo gradnika *RelativeLayout*, znotraj katerega smo uporabili gradnike tipa *Button* (gumbi).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="neki.poziciraj3.test.myapplication.Main3Activity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="32dp"
        android:text="Sledi mi"
        android:id="@+id/button2"
        android:layout_marginTop="60dp"
        android:layout_centerHorizontal="true"
        android:onClick="slediMi"/>

    <Button
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Zemljevid"
        android:textSize="32dp"
        android:id="@+id/button3"
        android:layout_below="@+id/button2"
        android:layout_alignLeft="@+id/button2"
        android:layout_alignStart="@+id/button2"
    android:layout_marginTop="150dp"
    android:onClick="sledimZemljevid"/>

</RelativeLayout>
```

V zgornjem primeru smo dva gradnika tipa *Button* prilagodili širini starševskega gradnika, ki je v tem primeru *RelativeLayout* in višino gradnika glede na velikost njegove vsebine. Besedilo znotraj gumbov smo določili s atributom *text* (Sledi mi), velikost pisave pa s pomočjo atributa *textSize*. Vsak gradnik v aplikaciji ima svoj identifikator (atribut *id*) preko katerega lahko manipuliramo gradnik v programskem jeziku Java. Atribut *onClick* nam omogoča, da s pritiskom na gumb izvedemo klic neke metode v programskem jeziku Java. Dodatno smo pozicionirali položaj gradnikov glede na višino in sredinjenje gradnika s pomočjo *layout_marginTop*, *layout_centerHorizontal, layout_below* itd.

S klikom na gumbe *Sledi mi* in *Zemljevid* poženemo njuni aktivnosti.

Ob zagonu aktivnosti razreda *SlediMi*, se nam odpre aktivnost, kjer imamo gradnike tipa *Button*, ki nam omogočajo zagon oddajanja trenutnih koordinat in zaključitev oddajanja teh koordinat.

Ko uporabnik pritisne na gumb za zagon oddajanja podatkov o lokaciji, se zažene dialogno okno (Slika 3.4), ki obvesti uporabnika, da bo s pritiskom na gumb za potrditev zagnal oddajanje svoje lokacije. Prav tako se mu izpiše identifikacijska številka, s katero lahko drugi uporabniki dostopajo do informacij o oddanih koordinatah.

Slika 3.4: Dialogno okno za oddajnik (različica za Android).

Takšno dialogno okno smo sestavili v dveh delih:

- Izdelali smo novo datoteko s strukturo jezika XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="1">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="32dp"
        android:id="@+id/usernameID"
        android:text="Uporabniški ID:"
        android:layout_gravity="center_horizontal"
        android:textColor="#ff0000"/>
</LinearLayout>
```

To nam omogoča prikaz besedila (gradnik *TextView*) znotraj dialognega okna, ki ga smo ga lahko, preko atributa z imenom *usernameID,* poljubno manipulirali s pomočjo programskega jezika Java.

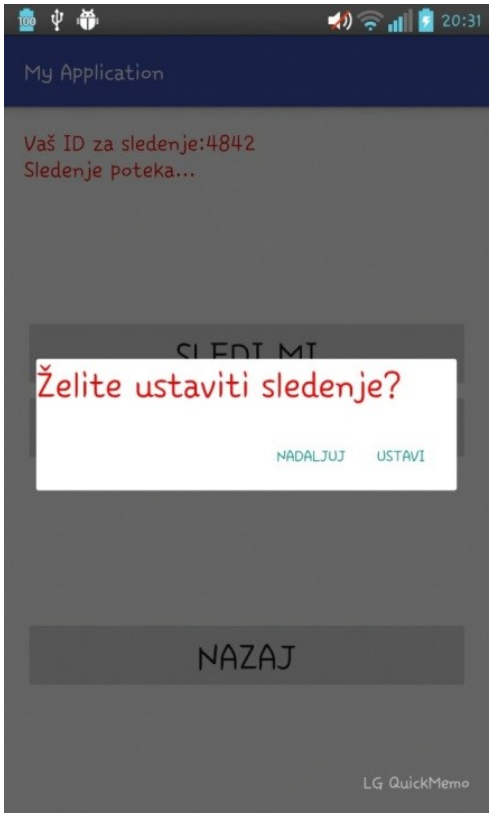
- V razredu *SlediMi* smo napisali metodo *pokaziDialog*, ki omogoča kreiranje dialognega okna s gumbom *ZAČNI* za potrditev in gumbom *PREKLIČI* za preklic dejanja:

```
public void pokaziDialog(String idUser) {
    AlertDialog.Builder dialog = new
AlertDialog.Builder(this);
    LayoutInflater inflater = this.getLayoutInflater();
    View dialogView = inflater.inflate(R.layout.dialog_sledi,
null);
    dialog.setView(dialogView);
    TextView e = (TextView)
dialogView.findViewById(R.id.usernameID);
    e.setText("Vaš ID za sledenje je: "+idUser);
    dialog.setPositiveButton("ZAČNI", new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int
kateriGumb)        {
            //OB PRITISKU NA 'ZAČNI' ZAŽENEMO SLEDENJE.
        }
    });
    dialog.setNegativeButton("PREKLIČI", new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int
kateriGumb) {
           //PREKLIC DIALOGA
        }
    });
    AlertDialog b = dialog.create();
    b.show();
}
```

Kot lahko vidimo v zgornji metodi, smo za kreiranje dialognega okna uporabili objekt tipa *AlertDialog*, ki je del operacijskega sistema Android. Objekt tipa *AlertDialog* ima metodi *setPositiveButton*, ki nam omogoča nastavitev potrditvenega gumba (nastavili smo mu ime *začni* in poslušalec za pritisk na gumb) ter *setNegativeButton*, ki smo ga nastavili za preklic storitve.

Razreda *LayoutInflater* in *View* nam omogočata dostop do datoteke s imenom *dialog_sledi*, kjer smo definirali strukturo vsebine dialognega okna. Do gradnika, kjer izpisujemo identifikacijsko številko pa smo dostopali s objektom tipa *TextView*, ki smo mu podali informacije o gradniku preko njegovega identifikatorja *usernameID*. S metodo *setText* smo nato določili izpisano vsebino v gradniku (identifikacijska številka oddaljene mobilne naprave).

Na enak način smo naredili tudi metodo *ustaviSledenje*, s katero zaključimo oddajanje koordinat (Slika 3.5). Razlika je samo v vrsti izpisa in vsebina znotraj potrditvenega gumba (zaključimo sledenje).



Slika 3.5: Dialogno okno za zaustavitev oddajnika (različica za Android).

Programska koda (funkcionalnost)

Funkcionalnost aplikacije, oziroma obnašanje aplikacije ob pritisku na gumbe, deluje na podlagi programskega jezika Java. S njegovo pomočjo imamo možnost manipulacije elementov XML in manipulacije obnašanja celotne aplikacije. Za večino funkcionalnosti lahko uporabljamo obstoječe razrede, ki so del operacijskega sistema Android.

Tako se med aktivnostmi pomikamo s pomočjo razreda *Intent*, ki omogoča zagon poljubne aktivnosti:

```
public void slediMi(View view) {
    Intent slediMi = new Intent(getApplicationContext(),
SlediMi.class);
    startActivity(slediMi);
}

public void sledimZemljevid(View view) {
    Intent zemlevid = new Intent(getApplicationContext(),
Zemljevid.class);
    startActivity(zemlevid);
}
```

Zaznavanje geografskega položaja oddaljene mobilne naprave

Za določanje geografskega položaja oddaljene mobilne naprave, lahko uporabljamo več načinov:

- GPS,
- internetno omrežje.

V našem primeru smo za zaznavanje koordinat uporabili GPS, ki nam omogoča zaznavanje trenutnega geografskega položaja preko satelitov.

Za uporabo sistema GPS smo uporabili objekt tipa *LocationManager*, ki nam omogoča periodično posodabljanje podatkov o naši lokaciji.

```
LocationManager locationManager =
(LocationManager)getSystemService(LOCATION_SERVICE);
```

Za periodično posodabljanje koordinat s pomočjo sistema GPS poženemo, smo uporabili metodo *requestLocationUpdates*, ki potrebuje 4 argumente:

- ponudnika storitev (GPS, internetno omrežje...)
- čas v mikrosekundah, ki določa, na koliko časa se naj podatki obnavljajo
- razdalja v čevljih, ki določa, na koliko čevljev se podatki obnovijo

- objekt tipa *LocationListener*, ki vsebuje metode, za zaznavanje sprememb lokacije (npr.: *onLocationChanged*).

V našem primeru smo tako napolnili metodo s podatki za uporabo sistema GPS, podatke obnavljamo na 1 minuto in ob vsaki spremembi položaja (npr.: 0 čevljev):

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
, 6000, 0, locationListener).
```

Za zaustavitev delovanja storitev GPS uporabimo metodo *locationManager.removeUpdates(locationListener)*.

Kot videno zgoraj potrebujemo za delovanje objektov tipa *LocationManager* tudi objekt tipa *LocationListener*. Ta nam ponuja metode:

- **onLocationChanged** - zaznava spremembo lokacije in vsebuje spremenljivko location (tip Location), ki vsebuje koordinati višine (ang.: latitude) in širine (ang.: longitude). Koordinati dobimo s metodoma *location.getLatitude()* in *location.getLongitude ()*.
- **onStatusChanged** – zaznava spremembo stanja ponudnika storitev (npr.: GPS).
- **onProviderEnabled** – zaznava, če je ponudnik storitev (npr.: GPS) vključen.
- **onProviderDisabled** – zaznava, če jee ponudnik storitev (npr.: GPS) izključen.

Da lahko drugi uporabniki prejmejo koordinate o naši lokaciji, moramo naše koordinate poslati na spletni server, kjer se podatki obdelujejo in hranijo v podatkovni bazi.

To smo storili s pomočjo razreda *PosljiPodatke*, ki smo ga izpeljali iz razreda *AsyncTask*. Podedovali smo 2 metodi:

- **doInBackground** – (dedujemo obvezno) se izvede v ozadju asinhrono. Tukaj pošljemo podatke na strežnik in prejmemo odgovor s strani tega strežnika:

```
protected String doInBackground(String...urls) {
    HttpURLConnection povezava = null;
    try {
        String urlParameters="idUporabnika=" +
URLEncoder.encode(idUporabnika, "UTF-8");
        if(!idUporabnika.equals("0")){
                urlParameters = "latitude=" +
```

```
URLEncoder.encode(String.valueOf(latitude), "UTF-8") +
                                    "&longitude=" +
URLEncoder.encode(String.valueOf(longitude), "UTF-8") +
                                    "&idUporabnika=" +
URLEncoder.encode(idUporabnika, "UTF-8");
        }
            //ustvari povezavo preko domene, ki je navedena v
urls[0]
            URL url = new URL(urls[0]);
            povezava = (HttpURLConnection)
url.openConnection();
            povezava.setRequestMethod("POST");
            povezava.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
            povezava.setRequestProperty("Content-Length", ""
+ Integer.toString(urlParameters.getBytes().length));
            povezava.setRequestProperty("Content-Language",
"en-US");
            povezava.setUseCaches(false);
            povezava.setDoInput(true);
            povezava.setDoOutput(true);

            //pošlji podatke
            DataOutputStream wr = new
DataOutputStream(povezava.getOutputStream());
            wr.writeBytes(urlParameters);
            wr.flush();
            wr.close();

            //prejmi odgovor
            InputStream is = povezava.getInputStream();
            BufferedReader rd = new BufferedReader(new
InputStreamReader(is));
            String line;
            StringBuffer odgovor = new StringBuffer();
            while ((line = rd.readLine()) != null) {
                odgovor.append(line);
            }
            return odgovor.toString();
    } catch (Exception e) {
        return null;
    } finally {

        if (povezava != null) {
            povezava.disconnect();
```

```
            }
        }
    }.
```

V zgornjem primeru smo ustvarili povezavo z imenom *povezava,* preko katere smo dostopali do našega spletnega strežnika. Nastavili smo metodo, da programski jezik na strani serverja prejema podatke kot spremenljivko *POST*. To nam omogoča večjo varnost poslanih podatkov. Parametre, ki jih pošiljamo so združeni v niz tipa *String* in sicer pošiljamo višino, širino in id uporabnika v nizu *urlParameters.*

- **onPostExecute** – (dedujemo po potrebi) se izvede, ko se metoda *doInBackground* zaključi. Tukaj tudi obdelamo podatke, ki jih prejmemo nazaj s serverja.

Razred *PosljiPodatke* se nato uporabi znotraj metode *onLocationCanged*, ki se izvede vsakič, ko spremenimo našo lokacijo:

```
public void onLocationChanged(Location location) {
    if (locationManager != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();
        PosljiPodatke id = new PosljiPodatke();
        id.execute(new String[]{"http://www.naša-
domena.si/prejem.php"});
    }
}
```

Prejemnik

Prejemnik je del aplikacije, ki nam omogoča prejemanje podatkov o geografskem položaju oddaljene mobilne naprave in vizualni prikaz te lokacije na zemljevidu (Slika 3.6). Uporabnik ima na voljo naslednje opcije:

- **Začni** – požene dialogno okno za vpis identifinacijske številke, preko katere iz baze pridobimo koordinate oddaljene mobilne naprave.
- **Končaj** – zaključi sledenje trenutni oddaljeni mobilni napravi.
- **Zgodovina** – prikaže zgodovino celotne poti oddaljene mobilne naprave, odkar smo ji začeli slediti (izriše točke in njihove medsebojne povezave, kar nam omogoča izris točne poti – v kolikor je oddaljena mobilna naprava oddajala koordinate dovolj pogosto).
- **Ustavi zgodovino** – nas vrne nazaj na sledenje trenutni oddaljeni mobilni napravi.

Slika 3.6: Grafični vmesnik za prejemnik (različica za Android).

Grafični vmesnik

Poleg gumbov, ki so gradniki tipa *Button*, se na tej aktivnosti nahaja gradnik tipa *Fragment*, ki nam omogoča prikaz zemljevida (sistem Google Maps).

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="80"
    android:layout_gravity="top" />
```

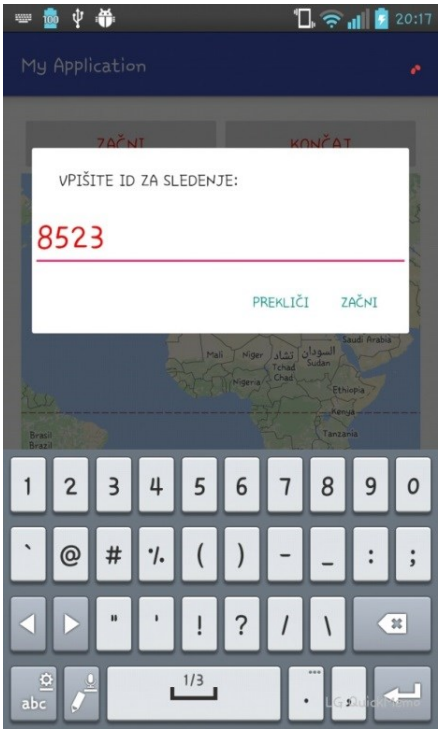
Zaradi uporabe sistema Google Maps, smo morali za atribut *name* določiti zgoraj napisano domeno. Ta domena omogoča uporabo fragmentov za prikaz zemljevidov. Alternativno bi lahko uporabili *MapFragment*, vendar je ta omogočen zgolj za novejše različice sistema Android, medtem ko *SupportMapFragment* deluje tudi na starejših različicah.

Ob pritisku na gumb *Začni,* se tako kot pri prej opisani različici za oddajanje koordinat, odpre dialogno okno (Slika 3.7), ki smo ga kreirali identično kot ostale različice, vendar tokrat ne uporabljamo gradnik *TextView,* ampak gradnik *EditText*, ki nam omogoča vnos besedila:

```
EditText e = (EditText) dialogView.findViewById(R.id.vpisiID);
```

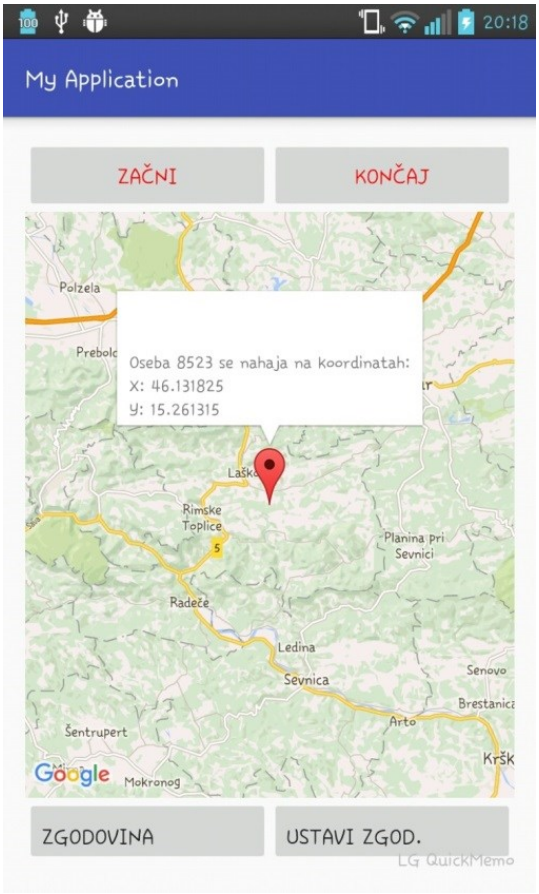
Slika 3.7: Dialogno okno za vnos identifikacijske številke (različica za Android).

Ob pritisku na prazno polje tega gradnika se nam avtomatsko odpre tipkovnica, s katero lahko tipkamo besedilo (Slika 3.8).



Slika 3.8: Dialogno okno za vnos identifikacijske številke s tipkovnico (različica za Android).

Po vpisu pravilne identifikacijske številke, se ob pritisku na gumb za potrditev, požene metoda *zacniSlediti*, ki sproži osveževanje in izris oznake (ang.: marker) na zemljevidu (oznaka prikazuje lokacijo uporabnika) (Slika 3.9). To osveževanje se sproža na programersko določen čas (priporočeno je osveževanje na 60 sekund ali več).



Slika 3.9: Izris geografskega položaja uporabnika (različica za Android).

Ko želimo prenehati slediti, lahko pritisnemo na gumb *Končaj*, s katerim se odpre dialogno okno s gradikom *TextView*, ki nam ob pritisku na gumb OK izpiše da se je sledenje končalo. Prav tako se stem tudi zaustavi zanka, ki nam osvežuje zemljevid.

Zemljevid (Google Maps)

Da lahko uporabljamo zemljevide sistema Google Maps, potrebujemo poseben ključ za avtentikacijo, ki smo ga pridobili na spletni strani https://console.developers.google.com.

Za pridobitev ključa smo se lahko prijavili s osebnim računom, ki smo ga registrirali za uporabo elektronske pošte pri podjetju Google. Ko smo ustvarili nov avtentikacijski ključ, smo ga za uporabo vstavili v datoteko *AndroidManifest.xml* v naslednji obliki:

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="Tukaj vstavimo avtentikacijski ključ" />.
```

Za testiranje povezave smo v programskem jeziku Java, uporabili razred *GooglePlayServicesUtil*, ki nam s metodo *isGooglePlayServicesAvailable* vrne podatek tipa intenger. Ta podatek smo zatem primerjali s rezultatom povezave, ki smo ga pridobili s pomočjo razreda *ConnectionResult.SUCCESS:*

```
int dosegljivo=
GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);
if(dosegljivo== ConnectionResult.SUCCESS){
    return true;
}
```

Inicializacija zemljevida

Naslednji korak uporabe Google Maps je bila inicializacija zemljevida, ki ga povežemo s gradnikom *fragment*.

Inicializacijo smo izvedli na naslednji način:

- določili smo globalno spremenljivko *mMap*, ki je tipa *GoogleMaps*
- Pridobili smo referenco na element fragment, ki je tipa *SupportMapFragment*:

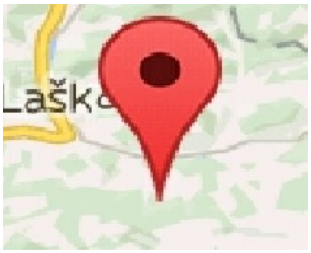
```
SupportMapFragment smf=
(SupportMapFragment)getSupportFragmentManager().findFragmentB
yId(R.id.map);
```

- Objektu *mMap* smo določili delovanje zemljevida na fragmentu s imenom *map*

```
mMap=smf.getMap();
```

Oznaka

Izris oznake na zemljevidu (Slika 3.10), nam omogoča vizualni prikaz geografskega položaja oddaljene mobilne naprave na zemljevidu.



Slika 3.10: Izris oznake na zemljevidu (različica za Android).

Za prikaz oznake smo napisali metodo *pokaziLokacijo*, ki nam omogoča prikaz označbe na zemljevidu s pomočjo koordinat (lat, lng) ter približanje kamere k tlom:

```
public void pokaziLokacijo(double lat,double lng, float zoom){
    CameraUpdate update= CameraUpdateFactory.newLatLngZoom(new
LatLng(lat, lng), zoom);
    mMap.animateCamera(update);
}
```

Zgoraj napisana metoda vsebuje objekt *update*, tipa *CameraUpdate*, ki osveži podatke o lokaciji na zemljevidu glede na koordinate, ki smo jih podali v objekt *new LatLng(lat, lng)*. Za dejanski premik oziroma animacijo na zemljevidu se uporabi metoda *mMap.animateCamera*, ki vizualno premakne našo prejšno lokacijo na novo lokacijo. Vsakič, ko uporabimo metodo *pokaziLokacijo*, se lokacija na zemljevidu spremeni glede na koordinate, ki smo jih podali kot argumente.

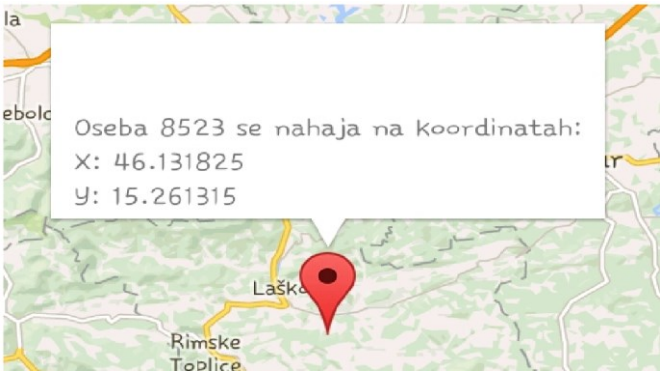
Za uporabo in prikaz *označbe* smo uporabili objekt *mark*, ki je tipa *Marker*.

```
public Marker mark;

public void pokaziLokacijo(double lat,double lng, float zoom){
    CameraUpdate update= CameraUpdateFactory.newLatLngZoom(new
LatLng(lat, lng), zoom);
    if(mark!=null) {
        mark.remove();
    }
    mMap.animateCamera(update);
    MarkerOptions opt = new
MarkerOptions().position(ll).icon(BitmapDescriptorFactory.defaultMa
rker(BitmapDescriptorFactory.HUE_RED));//za sličico uporabimo
icon(BitmapDescriptorFactory.fromResource(R.drawable.ic_puscica));
    mark = mMap.addMarker(opt);
    mark.showInfoWindow();
}
```

Oznako smo dodali na zemljevid s pomočjo metode *mMap.addMarker*, ki potrebuje za argument objekt tipa *MarkerOptions*. S pomočjo objekta, tipa *MarkerOptions,* lahko določimo lokacijo, kjer se naj izriše oznaka, prav tako pa lahko tej oznaki določimo tudi barvo ali pa jo nadomestimo s drugo sličico/ikono, ki smo jo predhodno shranili v mapo *drawable*. S metodo *mark.infoWindow* smo prikazali posebno okno nad oznako (Slika 3.11), v kateri

lahko podamo dodatne informacije o tej oznaki (npr.: koordinate te lokacije, ime oddaljene mobilne naprave, ki ji sledimo itd...).



Slika 3.11: Izris informacijskega okna s vsebino koordinat (različica za Android).

Informacijsko okno

InfoWindow je metoda, ki nam omogoča prikaz okna nad *oznako*. Obliko in vsebino informacijskega okna lahko določimo v dveh delih:

- Izdelamo novo datoteko v jeziku XML, kjer s pomočjo elementov *TextView* določimo obliko in količino besedil.
- V metodi, kjer smo inicializirali zemljevid, določimo novo prikazno okno s pomočjo metode *smMap.setInfoWindowAdapter*, kjer določimo podatke, ki jih naj okno prikazuje:

```
if(mMap!=null){
    mMap.setInfoWindowAdapter(new
GoogleMap.InfoWindowAdapter(){
        @Override
        public View getInfoWindow(Marker marker) {
            return null;
        }
        @Override
        public View getInfoContents(Marker marker) {
            View v=
getLayoutInflater().inflate(R.layout.costume_info_window,null
);
            TextView infoNaslov= (TextView)
v.findViewById(R.id.naslovinfo);
            TextView infoOseba = (TextView)
```

```
v.findViewById(R.id.oseba);
            TextView infoLat= (TextView)
v.findViewById(R.id.latitude);
            TextView infoLon= (TextView)
v.findViewById(R.id.longitude);
            infoNaslov.setText(marker.getTitle());
            infoOseba.setText("Oseba "+id+" se nahaja na
koordinatah:");
            infoLat.setText("X: "+latitude);
            infoLon.setText("Y: "+longitude);
            return v;
        }
    });
}
```

V zgornjem primeru imamo tako določeno, da naj se v prikaznem oknu izpisuje naslov, identifikacijska številka oddaljene mobilne naprave ter dolžina in širina, kjer se ta oddaljena mobilna naprava nahaja.

Prejemanje koordinat

Podobno kot pri oddajanju koordinat, smo tudi tukaj uporabljali asinhrono komuniciranje s strežnikom. Izpeljali smo podrazred *PrejmiKoordinate*, ki je podrazred razreda *AsyncTask*. Za razliko od *oddajnika*, tokrat nismo pošiljali uporabniškega imena in koordinat ampak samo uporabniško ime za avtentikacijo:

```
String urlParameters = "uporabniskiId=" + URLEncoder.encode(id,
"UTF-8");
```

Za razliko od prej je tokrat bila metoda *onPostExecute* nujna, saj smo skozi njo prejemali podatke o koordinatah:

```
@Override
protected void onPostExecute(String bl) {
    bl=bl.substring(1,bl.length()-1);
    String[] spl = bl.split(";");
    latitude=Double.parseDouble(spl[0]);
    longitude=Double.parseDouble(spl[1]);
    gotolocation(latitude,longitude, 10);
}
```

Prejete koordinate, ki smo jih uporabili kot niz podatkov tipa *String*, smo zapisali v spremenljivke latitude in longitude, v obliki tipa *double*. Te spremenljivke smo nato uporabili kot argumente metode *pokaziLokacijo*, ki nam prikaže novo lokacijo in *oznako* na zemljevidu.

Ker želimo prejemati podatke o koordinatah ves čas od pritiska na gumb za začetek, do pritiska na gumb za konec, smo za pošiljanje zahtev za koordinate, morali izvajati zanko, ki se izvaja na omejen čas. To smo dosegli s pomočjo razredov *Handler* in *Runnable*.

```
Handler hand = new Handler();
Runnable runHad = new Runnable()
{
    @Override
    public void run() {
        PosljiPodatke() vstavi = new PosljiPodatke();
        vstavi.execute(new String[]{"http://www.moja-
domena.si/koordinate.php"});
        hand.postDelayed(runHad, 5000);
    }
};
```

V zgornjem primeru se pošiljanje izvaja na vsake 5 sekund. Če želimo pognati zanko pokličemo metodo *runHad.run*.

```
public void zacniSlediti(View view){
    runHad.run();
}
```

Če želimo prekiniti izvajanje zanke moramo uporabiti metodo *had.removeCallbacks(runHad)*, ki se sproži s pritiskom na gumb za zaključek pridobivanja koordinat:

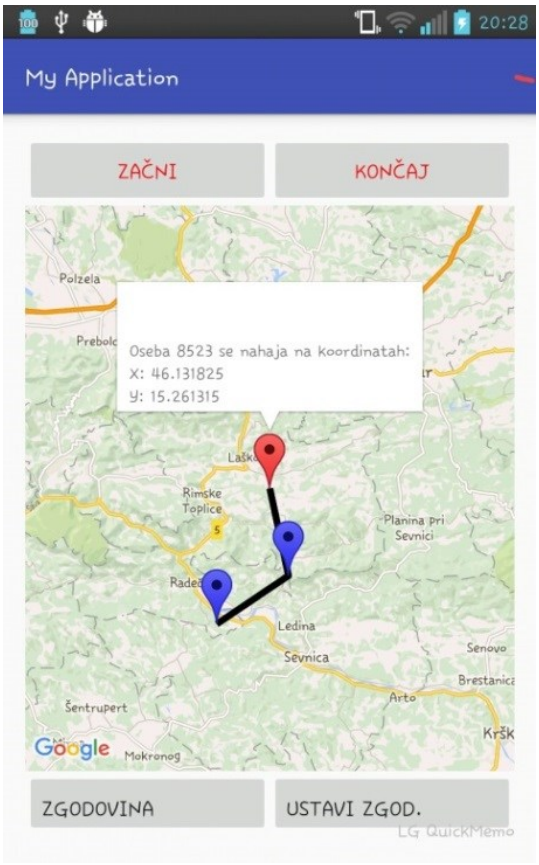
```
public void koncajSledenje(View view) {
    had.removeCallbacks(runHad);
    Toast.makeText(this,"Sledenje
končano!",Toast.LENGTH_SHORT).show();
}
```

Zgodovina sledenja

Gumb za prikaz zgodovine nam omogoča, da vidimo zadnje lokacije, kjer se je oddaljena mobilna naprava nahajala, odkar smo ji začeli slediti. Omogoča nam tudi izris povezav med lokacijami. Ta opcija omogoča, da če oddaljeni mobilni napravi sledimo določen neprekinjen čas, dobimo natančen izris poti te oddaljene mobilne naprave.

```
public void zgodovina(View view) {
    koncajSledenje(view);
    PolylineOptions po= new PolylineOptions();
    for(Marker m : oznake){
        MarkerOptions op = new
MarkerOptions().position(m.getPosition()).icon(BitmapDescriptorFact
ory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
        mMap.addMarker(op);
        po.add(m.getPosition());
    }
    mMap.addPolyline(po);
}
```

Pred izvedbo izrisa zgodovine poti se mora trenutno sledenje končati. Izris zgodovine poti (Slika 3.12) nam omogoča razred *PolylineOptions*, oznake pa izrišemo s pomočjo zanke. Oznake za izris zgodovine smo hranili znotraj objekta tipa *ArrayList*. Dodajali smo jih vsakič, ko smo izvedli spremembo lokacije, s pomočjo metode *oznake.add(mark)*. Zaradi lažjega ločevanja med zadnjo lokacijo in pretekle lokacije smo spremenili barvo oznake v modro.



Slika 3.12: Izris zgodovine poti uporabnika (različica za Android).

Alternativno bi lahko podatke o lokacijah pridobili iz podatkovne baze in dobili celotno zgodovino vseh lokacij, kjer se je oddaljena mobilna naprava nahajala.

3.3.3 Spletna različica aplikacije

Spletne strani sestavljamo s pomočjo kombinacije označevalnih, opisnih ter programskih jezikov. Tako s jezikom HTML podamo strukturo spletne strani (npr.: gumbi, opisna polja, sekcije...), s jezikom CSS oblikujemo elemente HTML, s programim jezikom Javascript določimo obnašanje spletne strani ter s programskim jezikom PHP, ki nam omogoča delovanje spletne strani v ozadju (npr.: obdelava podatkov, povezava in delo s podatkovno bazo...).

Tako kot pri prejšni različici naše aplikacije, imamo tudi tukaj 2 opciji za uporabo aplikacije:

- Opcija za oddajanje koordinat, kjer se mobilna naprava nahaja (oddajnik).
- Opcija za prejemanje koordinat in prikaz na zemljevidu, kje se oddaljene mobilne naprave nahaja (prejemnik).

Oddajnik

Tako kot pri različici za Android je tudi tukaj oddajnik je del aplikacije, ki omogoča oddajanje podatkov o trenutnemu geografskemu položaju mobilne naprave (Slika 3.13). Uporabnik ima na voljo naslednje opcije:

- Začni oddajati koordinate.
- Zaključi oddajanje koordinat.

Slika 3.13: Grafični vmesnik za oddajnik (spletna različica).

Grafični vmesnik

Uporabniški vmesnik na spletni strani sestavljajo elementi jezika HTML, ki jih dodatno vizualno oblikujemo s pomočjo jezika CSS:

```
<body>
	<div id="sekcija">
		<button onclick="odpriDialog()">ZACNI
SLEDENJE</button><br/>
		<button onclick="ustaviZanko()">KONCAJ SLEDENJE</button>
		<span id="sledenjePoteka">Sledenje poteka...</span>
	</div>
</body>
```

V zgornjem primeru smo tako določili 2 gumba tipa *button*, ki nam omogočata zagon funkcij *odpriDialog* in *ustaviZanko*, napisanih v programskem jeziku Javascript. Dodatno smo vključili tudi element *span*, ki nam omogoča izpis opisa dogajanja ob vklopu sledenja.

V jeziku CSS smo nato dodatno oblikovali zgornje elemente:

```
html, body{height: 100%; width: 100%;}
#sekcija{
	margin: auto;
	margin-top: 100px;
	width: 50%;
```

```
      padding: 10px;
}
button{
      width: 100%;
      color: red;
      height: 60px;
}
```

Programska koda (funkcionalnost)

Za odpiranje dialognih oken, pošiljanja podatkov na strežnik in določanja trenutnega geografskega položaja oddaljene mobilne naprave, smo uporabili spletni programski jezik Javascript. Programski jezik Javascript vsebuje vgrajena dialogna okna kot je *alert* za objavo opozoril uporabniku, *confirm dialog* za potrditveno okno in ostalo. V našem primeru smo za potrjevanje sledenja uporabili potrditveno okno (angleško *confirm dialog*).

```
function odpriDialog() {
    narediID(); //ustavri identifikacijsko številko
    var r = confirm("ID ZA SLEDENJE: "+ idUser);
    if (r == true) {
        prodobiPozicijo();
    }
}
```

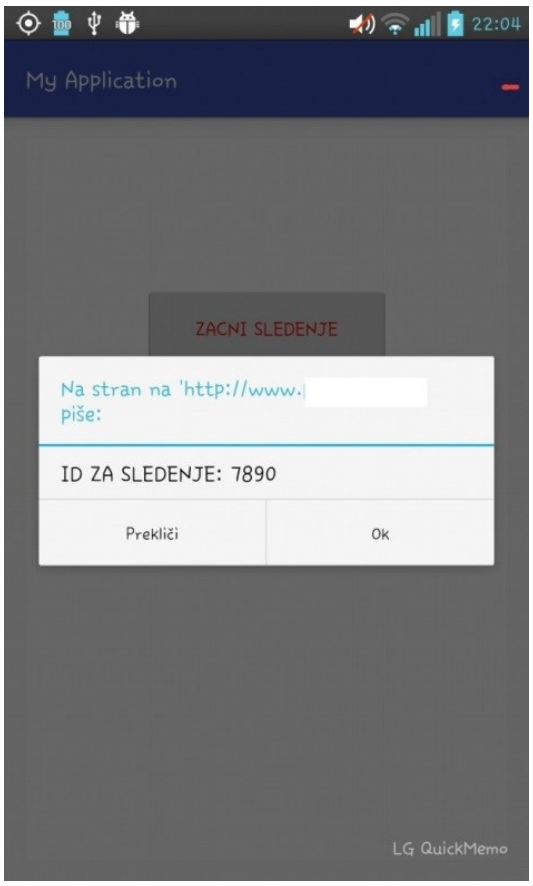
Ob odprtju dialognega okna se požene funkcija s imenom *narediID*, ki preko sistema Ajax, pošlje na strežnik zahtevo za ustvaritev nove identifikacijske številke za mobilno napravo:

```
function narediID(){
      $.ajax({
            url: '/skripta.php',
            dataType: 'json',
            type: "POST",
            async: false,
            data:{
                  'idUporabnika':idUser,
            },
            success:function(data) {idUser=data; }
      });
}
```

V zgornjem primeu smo tako s strani strežnka prejeli podatek preko argumenta *data*, ki smo ga shranili v spremeljivko *idUser*. Spremenljivko *idUser* lahko nato uporabljamo za identifikacijo mobilne naprave, ko želimo pridobivati informacije o njenih koordinatah.

Sistem Ajax se v osnovnem načinu delovanja, izvaja asinhrono, vendar smo ga za naše potrebe prilagodili, da se izvaja sinhrono. To smo dosegli s pomočjo *async:false*. Podatke pošiljamo v obliki JSON. Vsi podatki se pošiljajo preko povezave na isto datoteko *prejem.php*, ki smo jo uporabili tudi v prejšni različici aplikacije za sistem Android.

Po odprtju dialognega okna (Slika 3.14) imamo na voljo gumb za izbiro potrditve zagona sledenja, ki požene funkcijo *pridobiPozicijo*, ta pa zatem prodobiva informacije o trenutni lokaciji in jih pošilja na strežnik.



Slika 3.14: Dialogno okno za oddajnik (spletna različica).

Navigator

Programski jezik Javascript vsebuje poseben objekt po imenu *Navigator*. S pomočjo navigatorja, lahko pridobimo različne informacije o brskalniku, kot je informacija o platformi, delovanju programskega jezika Java, uporabi piškotkov, geolokaciji in podobno.

Za nas je bila pomembna geolokacija, ki smo jo prodobili s funkcijo*:*

```
function pridobiPozicijo() {
```

```
        if (navigator){
              if(navigator.geolocation){
              opcije = { enableHighAccuracy: true, timeout: 5000,
              maximumAge: 0};
              navigator.geolocation.watchPosition(izvediPrenos,napak
              a,opcije)
              }
        }
}
```

S funkcijo *navigator.geolocation.watchPosition* smo določili, da naj program pridobi podatke o geografskem položaju ob vsakem premiku. Funkcija *watchPosition* ima na voljo 3 argumente:

- Klic funkcije, ki se izvede ob vsaki spremembi geografskega položaja (tukaj uporabimo podatke o dolžini in širini ter izvedemo prenos podatkov na server preko sistema Ajax):

```
function izvediPrenos(position) {
     $.ajax({
        url: '/skripta.php',
        dataType: 'json',
        type: "POST",
       async: false,
      data:{
          'idUporabnika':idUser,
          'latitude': position.coords.latitude,
          'longitude': position.coords.longitude
   },
  success:function(data) {//prodibimo odgovor v obliki
argumenta data},
   });}.
```

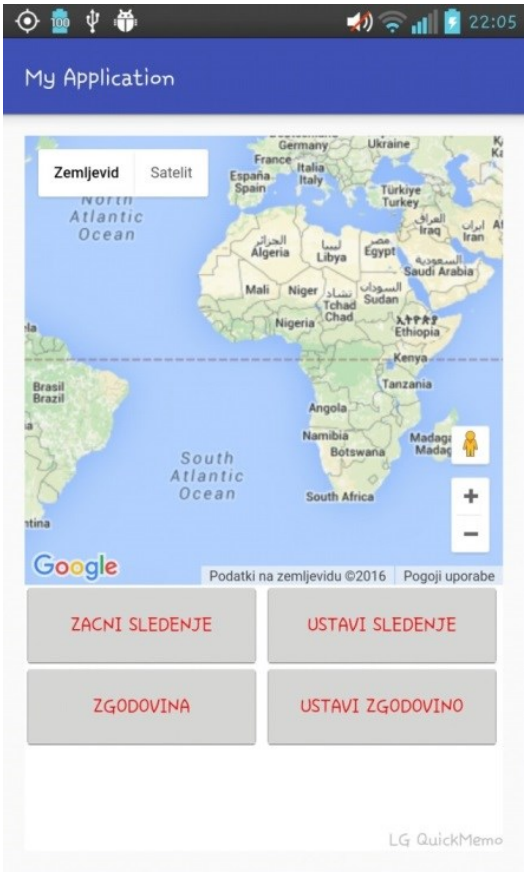
- Klic funkcije za napake.

- Uporaba dodatnih možnosti (*opcije*), kot je natančnost določanja geografskega položaja na čevelj natančno, čas določitve geografskega položaja in starost informacij o geografskemu položaju.

Prejemnik

Enako kot pri različici aplikacije za sistem Android je *prejemnik* del aplikacije, ki omogoča uporabniku prejemanje podatkov o geografskemu položaju oddaljene mobilne naprave in vizualni prikaz tega geografskega položaja na zemljevidu (Slika 3.15). Uporabnik ima naslednje opcije:

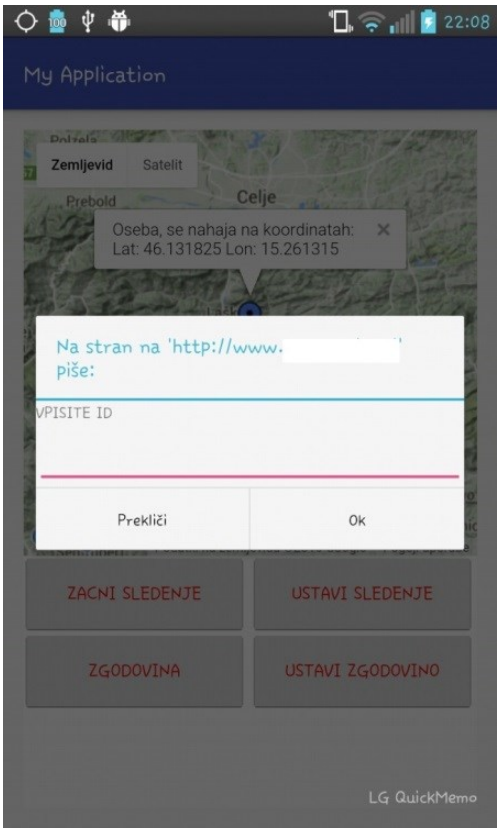
- **Začni** – požene dialogno okno za vpis identifinacijske številke, preko katere iz baze pridobimo koordinate oddaljene mobilne naprave.
- **Končaj** – zaključi sledenje trenutni oddaljeni mobilni napravi.
- **Zgodovina** – prikaže zgodovino celotne poti oddaljene mobilne naprave, odkar smo ji začeli slediti (izriše točke in njihove medsebojne povezave, kar nam omogoča izris poti).
- **Ustavi zgodovino** – nas vrne nazaj na sledenje trenutni oddaljeni mobilni napravi.



Slika 3.15: Grafični vmesnik za prejemnik (spletna različica).

Grafični vmesnik

Tako kot pri *oddajniku* smo tudi tukaj s elementi HTML sestavili skupino gumbov, ki smo jo vizualno uredili s jezikom CSS. Ob pritisku na gumb za začetek sledenja se odpre okno za vnos besedila (Slika 3.16), ki smo ga ustvarili s pomočjo vgrajenega dialognega okna *prompt*. Za zaustavitev sledenja smo ponovno uporabili vgrajeno dialogno okno *alert*.



Slika 3.16: Dialogno okno za vnos identifikacijske številke (spletna različica).

Programska koda (funkcionalnost)

Koordinate oddaljene mobilne naprave smo prejemali s pomočjo sistema Ajax, s katerim smo pošiljali uporabniško identifikacijsko število in prejemali odgovore strežnika s koordinatami. Za neprekinjeno pridobivanje podatkov smo tudi tukaj uporabili intervalno ponavljanje na določeno število sekund:

```
function dobiKoordinate(userId){
      interval = setInterval(function(){
            $.ajax({
                  url: '/skripta2.php',
                  dataType: 'json',
                  type: "POST",
```

```
                data:{'uporabniskiId':userId},
                success:function(data){
                     //preberi podatke iz data in jih zapiši v
spodnjo funkcijo
                     pokaziZemljevid(latitude, longitude, 15);
                },
           });}, 5000);}
```

Inicializacijo zemljevida

Izvedli smo jo s funkcijo *pokaziZemljevid*, ki prejme argumente za višino, širino in višinsko oddaljenost. Za začetno inicializacijo ob vklopu aplikacije smo tako vstavili funkcijo *pokaziZemljevid* v element HTML, ki s pomočjo funkcije *onload* aktivira funkcijo *pokaziZemljevid* ob naložitvi spletne strani. Nadaljno smo funkcijo *pokaziZemljevid* klicali intervalno in uporabljali koordinate kot je vidno v zgornjem primeru.

Prikazovanje novega geografskega položaja na zemljevidu se zaključi s pritiskom na gumb za zaključek sledenja, ki sproži funkcijo *clearInterval(interval),* ta pa zaustavi pridobivanje koordinat.

Funkcija *pokaziZemljevid*

Za uporabo zemljevida smo na spletni strani določili element *div*, ki smo mu določili identifikator s imenom *map*. Ko smo pridobili koordinate, smo jih shranili v spremenljivko *latLng*, s uporabo *google.maps.LatLng*, ki kot argumente sprejme koordinate in jih razvrsti v obliki tabele za kasnejšo uporabo.

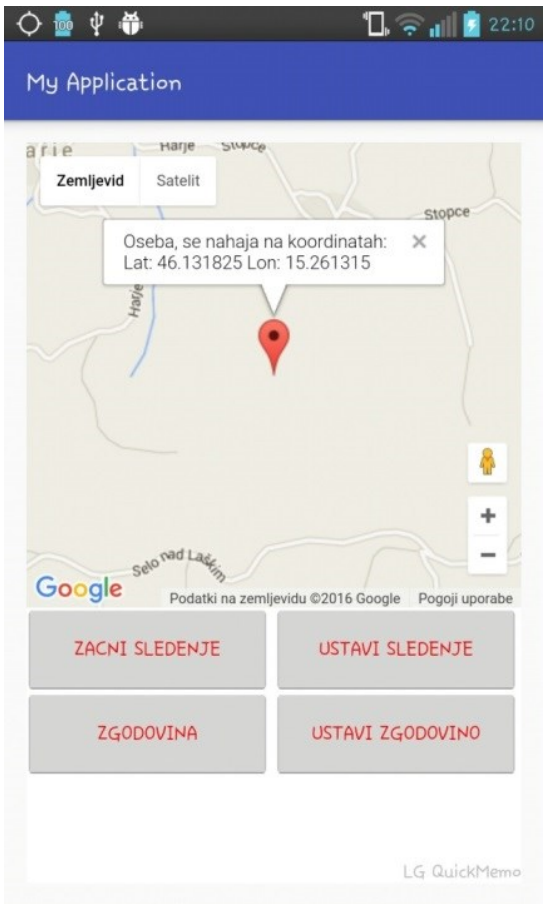
Za prikaz zemljevida in lokacije uporabnika, smo uporabili *google.maps.Map,* ki v elementu *div* prikaže zemljevid. V spremenljivko *opcije* smo shranili dodatne lastnosti, kot so koordinate prav tako smo vnesli podatek za višinsko oddaljenost in tip zemljevida:

```
function pokaziZemljevid(latitude, longitude, oddaljenost) {
     var latLng = new google.maps.LatLng(latitude, longitude);
      if(map == undefined) {
          var myOptions = {
            zoom: oddaljenost,
            center: latLng,
            mapTypeId: google.maps.MapTypeId.ROADMAP
          }
```

```
      var map = new google.maps.Map(document.getElementById("map"),
opcije);

            //tukaj upravljamo s oznakami
        }
        else map.panTo(latLng);
}
```

Funkcija *map.panTo* omogoča prikaz položaja na specifično lokacijo na zemljevidu (Slika 3.17).

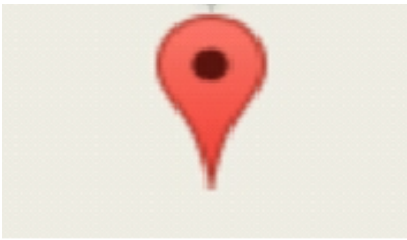


Slika 3.17: Izris geografskega položaja uporabnika (spletna različica).

Oznaka

Za manipulacijo oznak smo uporabili *google.maps.Marker*, ki nam omogoča določitev kje na zemljevidu se izrišejo oznake (Slika 3.18). Da se izrisuje samo oznaka, ki nam prikazuje trenutno lokacijo, smo morali odstraniti prejšne oznake s *marker.setMap(null):*

```
marker = new google.maps.Marker({
      position: latLng,
      map: map,
      title: 'zemljevid'
});
if (marker!=null){
      marker.setMap(null);
}
marker.setMap(map);
```

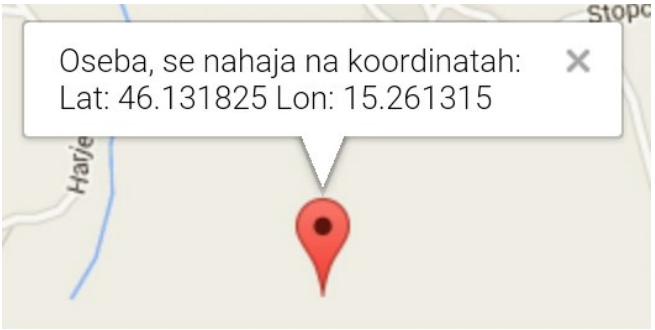


Slika 3.18: Izris oznake na zemljevidu (spletna različica).

Informacijsko okno

Sestavili smo ga s *google.maps.InfoWindow*, ki nam omogoča poljubno oblikovanje vsebine pojavnega okna, s pomočjo elementov HTML (Slika 3.19). Na koncu ga odpremo s funkcijo *open*.

```
var infowindow = new google.maps.InfoWindow({
                content: "Oseba, se nahaja na koordinatah: <br/>Lat:
"+latitude+"  Lon: "+longitude;
});
infowindow.open(map, marker);
```

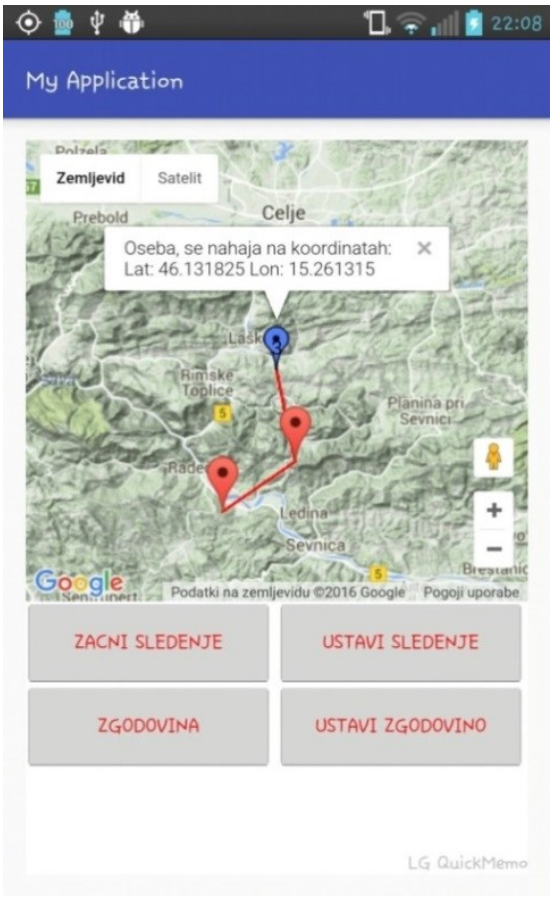


Slika 3.19: Izris informacijskega okna s vsebino koordinat (spletna različica).

Zgodovina

Da si uporabnik lahko pogleda zgodovino poti oddaljene mobilne naprave (Slika 3.20), smo podobno kot v prejšni različici aplikacije, shranili oznake v tabelo in jih nato izrisovali skozi zanko. Za izris poligonov smo uporabili *google.maps.Polyline*, ki omogoča izris poti med točkami. Dodatno je za pot možno določiti barvo in debelino črte kot lahko vidimo v primeru:

```
var pot = new google.maps.Polyline({
      path: tabelaMarker,
      geodesic: true,
      strokeColor: '#FF0000',
      strokeOpacity: 1.0,
      strokeWeight: 2
      });
pot.setMap(map);.
```



Slika 3.20: Izris zgodovine geografskih položajev oddaljene mobilne naprave (spletna različica).

4 Testiranje in primerjava

4.1 Domorodna različica

Domorodno aplikacijo smo testirali na mobilni napravi LG e975, kjer smo preiskusili zaznavanje geografskega položaja in oddajanje teh podatkov na spletni strežnik. Opazili smo, da je natančnost geografskega položaja zelo raznolika, saj potrebuje sistem GPS kar nekaj časa za natančno določitev našega geografskega položaja. Prav tako je natančnost zelo odvisna od samega vremena, naše lokacije in če se nahajamo znotraj ali izven neke stavbe.

Za testiranje izrisa zgodovine naše poti, smo testirali izris specifičnih koordinat, ki smo jih primerjali s že obstoječim sistemom Google Maps, ki nam ga ponuja podjetje Google na svoji spletni strani. Nismo opazili posebnih razlik med našo aplikacijo in aplikacijo, ki jo ponuja podjetje Google.

4.2 Spletna različica

Spletno različico aplikacije smo sprva testirali na računalniku, pri čemer smo test izvedli na operacijskem sistemu Windows 7 in brskalniku Firefox. Ker smo za testiranje na računalniku uporabili internet, smo dobili rahlo drugačne rezultate kot s sistemom GPS. Testiranje spletne različice smo zatem izvedli še na enaki mobilni napravi kot testiranje domorodne aplikacije. Rezultate smo primerjali med različicama testiranima na mobilni napravi in nismo opazili posebnih razlik v natančnosti določanja geografskega položaja, saj je natančnost določanja koordinat odvisen od samega sistema GPS. Tako je natančnost določanja geografskega položaja s sistemom GPS predvsem odvisna od naše lokacije, okolice, časa, ki ga porabimo za uporabo sistema GPS in vremena.

4.3 Primerjava

Prišli smo do zaključka, da je bila izdelava spletne različice aplikacije veliko bolj enostavna, vendar pa tudi bolj omejena pri možnostih manipulacije nekaterih delov, medtem ko je bila aplikacija izdelana izključno za platformo Android veliko bolj kompleksna za izdelavo,

vendar je ponujala več možnosti za manipulacijo vseh sestavnih delov. Prav tako je zaradi dostopnosti do strojne opreme domorodna različica aplikacije delovala hitreje kot spletna različica. Pomembno je tudi omeniti, da je bil čas izdelave spletne različice naše aplikacije veliko krajši kot čas izdelave aplikacije za sistem Android. V praksi je tako težavnost kot čas izdelave izredno velikega pomena, saj je izdelava kompleksnejših in dalj trajajočih aplikacij veliko dražje.

5 Zaključek

V tem diplomskem delu smo predstavili dve različici aplikacije, ki omogoča uporabniku vizualno predstavitev trenutnega geografskega položaja oddaljene mobilne naprave na zemljevidu. Izdelali smo različico aplikacije, ki deluje izključno na mobilnih napravah s operacijskim sistemom Android ter spletno različico te iste aplikacije, ki je neodvisna od platforme. V obeh primerih je aplikacija pošiljala in prejemala podatke s istega strežnika in njegovega sistema, ki je obdeloval podatke.

Orodje, ki smo ga uporabili za izdelavo domorodne aplikacije je bilo programsko okolje Android Studio, ki nam je omogočalo enostavno in pregledno izdelavo grafičnega vmesnika ter programske kode v ozadju. Za izdelavo spletne različice aplikacije, smo se odločili za uporabo beležke (ang.: notepad), za preiskušanje izgleda in delovanje aplikacije pa smo uporabljali spletni brskalnik Firefox.

Za pridobivanje geografskega položaja smo pri obeh različicah uporabili sistem GPS, za katerega smo ugotovili, da je njegova natančnost odvisna od okolice in same mobilne naprave. Posebnih razlik pri pridobivanju koordinat oddaljene mobilne naprave, med različicama nismo opazili. S primerjavo obeh različic smo tudi ugotovili, da je bila izdelava spletne različice aplikacije veliko bolj enostavna in hitrejša, vendar pa ima več omejitev s vidika programerskih možnosti. Prednost spletne različice je bila tudi neodvisnost od platforme, medtem ko je bila prednost domorodne različice aplikacije večja funkcionalnost, hitrejše delovanje in boljši grafični vmesnik.

Samo aplikacijo bi bilo možno nadgraditi in uporabljati za več različnih ciljev. Tako bi lahko prodobivali informacije o položaju prevoznikov storitev, pomoč učiteljem pri nadzoru nad otroci na ekskurzijah ali pa za izdelavo igric v stilu igre Pokemon GO.

Za večjo varnost in uporabnost bi se dodatno lahko uvedlo polno registracijo uporabnikov, s čimer bi uporabniško indentfikacijsko številko zamenjali s imenom in priimkom uporabnika. Namesto oznake bi se lahko izrisovala profilna sličica uporabnika, prav tako bi lahko omogočili uporabnikom celoviti pregled in nadzor nad zgodovino svojih geografskih

položajev (npr. pregled ali izbris včerajšnih koordinat). Možnosti za nadgradnjo in uporabo takšne aplikacije je vsekakor veliko. Prednost pri nadgradnji pa ima vsekakor različica aplikacije, ki je narejena izključno za platformo Android, saj imamo programerji nad njo večji nadzor, manj omejitev in več programerskih rešitev kot pa za spletno različico te iste aplikacije.

6 Viri in literatura

[1] (2016) Wikipedia, "Global Positioning System". Dostopno na: https://en.wikipedia.org/wiki/Global_Positioning_System.

[2] (2016) Wikipedija, "Internet". Dostopno na: https://sl.wikipedia.org/wiki/Internet.

[3] (2016)Wikipedia, "Client–server model". Dostopno na: https://en.wikipedia.org/wiki/Client–server_model.

[4] (2016) Wikipedia, "Google Maps". Dostopno na: https://en.wikipedia.org/wiki/Google_Maps .

[5] (2016) Wikipedia, "Website". Dostopno na: https://en.wikipedia.org/wiki/Website.

[6] (2016) Wikipedija, "HTML". Dostopno na: https://sl.wikipedia.org/wiki/HTML.

[7] (2016) Wikipedia, "Component-based software engineering". Dostopno na: https://en.wikipedia.org/wiki/Component-based_software_engineering.

[8] (2016) Wikipedia, "Mobile device". Dostopno na: https://en.wikipedia.org/wiki/Mobile_device.

[9] (2016) Wikipedia, "Tablet computer". Dostopno na: https://en.wikipedia.org/wiki/Tablet_computer.

[10] (2016) Wikipedia, "Smartphone". Dostopno na: https://en.wikipedia.org/wiki/Smartphone.

[11] (2016) Wikipedia, "Android". Dostopno na: https://en.wikipedia.org/wiki/Android_(operating_system).

[12] (2016) Wikipedia, "Android - zgradba operacijskega sistema". Dostopno na: https://sl.wikipedia.org/wiki/Android_(operacijski_sistem)#Zgradba_operacijskega_sistema.

[13] (2016) Wikipedia, " XML". Dostopno na: https://sl.wikipedia.org/wiki/XML.

[14] (2016) Wikipedia, "Java". Dostopno na: https://en.wikipedia.org/wiki/Java_(programming_language).

[15] (2016) Lars Vogel, "Introduction to Android development with Android Studio - Tutorial". Dostopno na: http://www.vogella.com/tutorials/Android/article.html#components_activity2.

[16] (2016) Elliotte Rusty Harold, "Java at 20: How it changed programming forever". Dostopno na: http://www.infoworld.com/article/2923773/java/java-at-20-how-java-changed-programming-forever.html.

[17] (2016) Martin Flower, "Writing with XML". Dostopno na: http://martinfowler.com/articles/writingInXml.html.

[18] (2016) Android developers, "Fundamentals". Dostopno na: http://developer.android.com/guide/components/fundamentals.html.

[19] (2016) Android developers, "SDK". Dostopno na: http://developer.android.com/sdk/index.html.

[20] (2016) Android developers, "Resources overview". Dostopno na: http://developer.android.com/guide/topics/resources/overview.html.

[21] (2016) Android developers, "User interface overview". Dostopno na: http://developer.android.com/guide/topics/ui/overview.html.